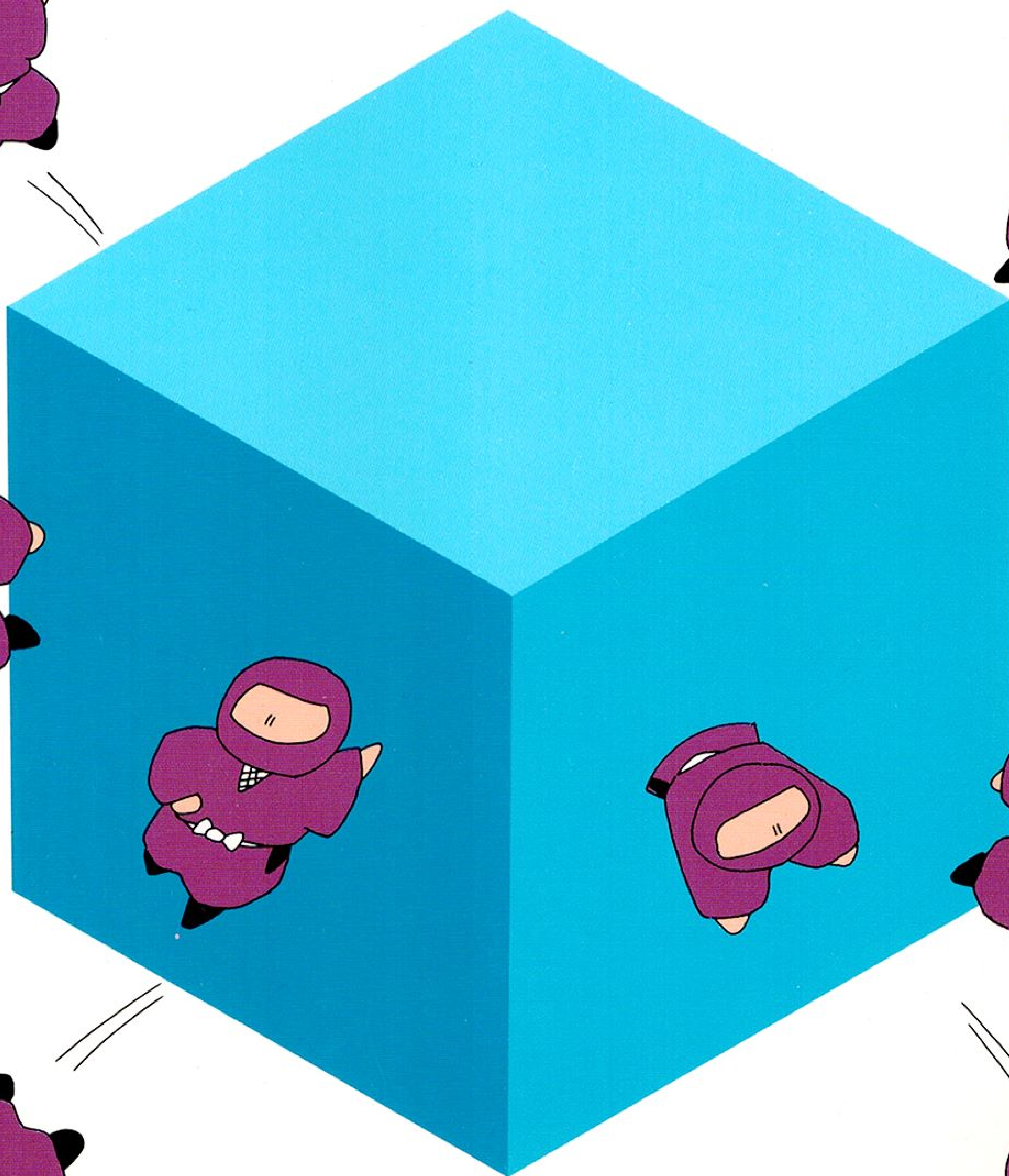
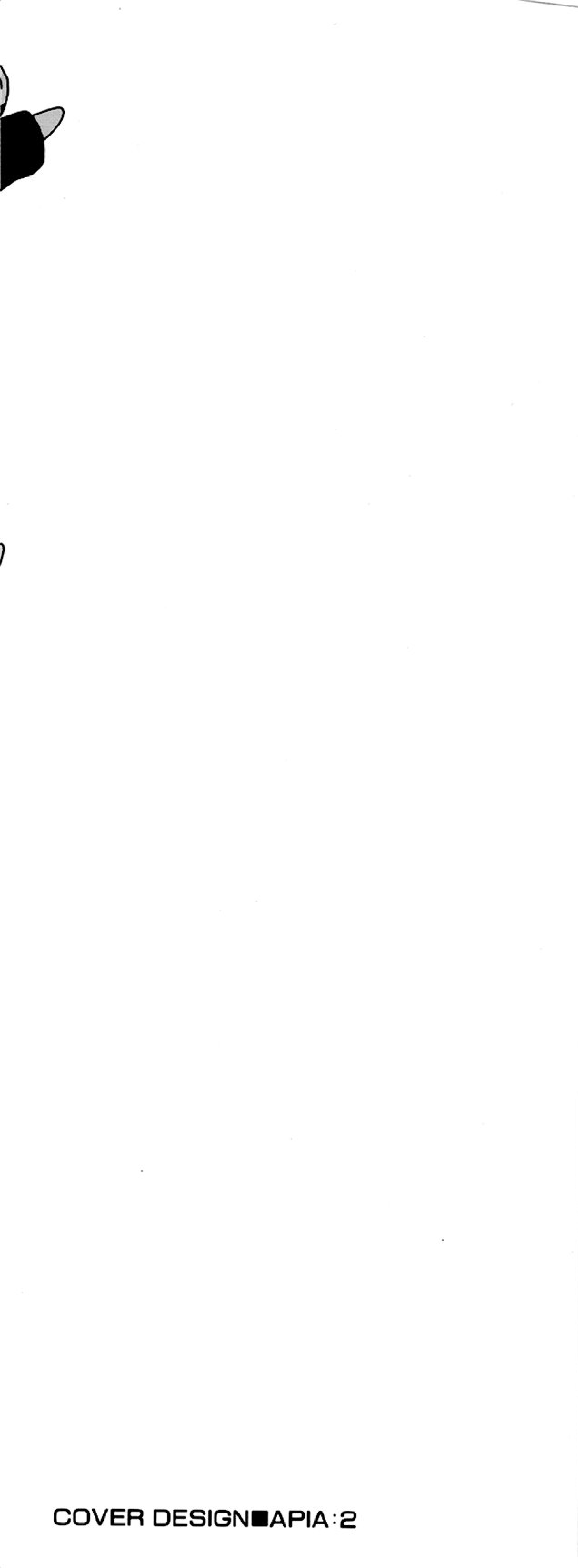
Z80 マシン語秘伝の書

■日高 徹 著



啓学出版



COVER DESIGN■APIA:2

Z80 マシン語秘伝の書

■日高 徹 著

啓学出版

はじめに

コンピュータ言語にも色々なものがありますが、1つだけすべてに共通していることがあります。それは、最終的には必ずマシン語になって、あるいはマシン語によって動いているという事実です。

フランス語、英語、スペイン語、日本語……。言葉に種類があるように、マシン語にもいくつかの種類があります。通常、ニモニックによるアセンブリ言語もマシン語といいますが、Z80、8080、8086、6502、6809、80286、80386 ……など、CPU の違いによってマシン語（ニモニック）はバラバラです。さらに、人間の言葉に方言があるように、マシン語にはハードウェアの違いという方言以上に面倒な壁が存在しています。

そのため、同じ Z80を搭載したコンピュータでも、実際の用法は機種によってまったく違うものとなります。その結果、CPU 別にマシン語の解説をしようとすると、どうしてもニモニックそのものの解説が中心になりがちです。また、機種別にマシン語を解説しようとすると、今度はハードウェア・コントロールに関する説明が必須となり、プログラムは急激に現実味を帯びたものになってしまいます。

そもそも、マシン語というものは複数の命令によって意味を成す言語ですから、ニモニックは単なるアルファベット的な存在に過ぎません。したがって、本来ならば基本的なアルゴリズムや基礎テクニックをマスターしてから、現実的なプログラムへと進むべきものです。しかし、それではマシン語に対する興味を持続することが困難となるので、とりあえずゲームなど楽しいことを目標にマシン語を覚えるわけです。

そこで、たとえ順序は逆であっても、ある程度マシン語をマスターしたなら、改めてゆっくりと基礎テクニックを振り返ってほしいのです。なぜなら、ハードウェアをコントロールすることはマシン語のほんの一端であり、その前後のプロセスにこそマシン語の真のテクニックがあるからです。

マシン語プログラムを組む人の中には、プログラムの無駄やアルゴリズムなど、どうせアセンブルすれば隠れてしまうという人もいます。これは、マシン語の基礎を省いてしまったことの弊害かもしれません。せっかく覚えたマシン語ですから、面白くないこともそれなりに知っておくことが大切なのです。

とはいえ、基礎テクニックが基礎のままではヤッパリ面白くありません。そこで、マンガを読むような気楽な気分で、つまらない基礎を知識の一部にしてしまいましょう。本書の基礎テクニックは、いわばプログラムの化粧品のようなもの。ちょっぴりプログラムにオシャレをするだけで、プログラムが見違えるほど光り輝くかもしれません。

いかにも機械的なマシン語ですが、アルゴリズムやテクニックというものには芸術に匹敵する'美'が存在しています。それを追求するかしないか、そのこと自体はプログラムを組む人の自由です。しかし、マシン語の技術レベルの差とは、案外そんなところに潜んでいるのではないでしょうか。

1989年7月　　日高　徹

目　次

一の章　秘伝の書を読む前に

本書は、世界各地から……いや存在さえ定かでない謎の世界からも寄せられた多くの質問の中から、Z80 の秘伝として後世に残すべき基礎テクニックばかりを集め、それに応える形で本にまとめ上げたマシン語の極意書であります。

その昔、日本には忍者という超能力にも似た秘術を使いこなす人間がいました。もちろん、忍者は一朝一夕になれるようなものではありません。常に死と背中合わせの厳しい修行を重ね、その中から自分の天分に見合った忍法を体得した者だけが、忍者として一流の称号を得ることができたのです。いくら厳しい修行に耐えようとも、オリジナルの忍法を開発できなかった者は、結局は三流の雑魚忍群として虫けらのような一生を送るしかなかったのです。

しかし、一流の忍者にも老化という力の衰える時期が必ずやってきます。その時、彼らは自分の秘術を残すために、その極意を巻物に記したといいます。こうして、秘術は巻物とともに後継者に受け継がれていくのです。もちろん後継者になれるのは、弟子の中から選ばれた、たった一人の超一流の忍者です。

……やがて、彼もその巻物に秘術を記す時が訪れます。こういった小さな歴史が代々繰り返されることにより、巻物はその流派の『秘伝忍法帖』、あるいは『秘伝の書』として無上の存在価値を持つようになるのです。つまり、「その巻物さえ手にいれれば、無条件で超一流の忍者になれる」という神話が出来上がるわけです。

こうなると、忍者であればその巻物が欲しくなって当然です。巻物を求めて、密告、裏切りが横行し、ついには疑心暗鬼から内ゲバへと発展、修行を忘れた凄惨な争いが起こるようになるのです。その結果、多くの忍者の里が闇から闇へと自滅していったのです。たった１つの巻物のために……。

それほどまでに人の心を狂わす『秘伝の書』……。その内容は、実は秘術を記したものではなかったのです。そこには、精神的な極意、つまり平常心を養うことの大切さや秘訣が抽象的に記されていただけでした。

忍の道を極めた者にとって、それこそがあらゆる忍法の極意であり、そこか

ら完成された秘術など一代限りのどうでもいい枝葉末節のことだったのです。厳しい修行はそれを悟るための手段であり、秘伝とはそういった悟りの境地を伝えることなのです。

すなわち、『秘伝の書』の価値はそれに見合う修行を積んだ者にのみ理解され、修行中の者には文面通りの内容しか伝わらないのです。例えば、部屋にある敷居の上は誰でも歩くことができますが、千尋の谷の上に架けられた同じ幅の木の上を平然と歩くことができるでしょうか。誰にでもできる簡単なことを、環境を選ばず冷静に実行できるようになるには、たゆまぬ基礎訓練が必要なのです。その中から自分の才能に適した新しいものを発見した時、オリジナルの秘術が生まれるのです。そして、その精神を平易に教えてくれるのが『秘伝の書』というわけです。

マシン語でプログラムを組めるようになると、どうしてもアッと驚く秘術ばかりを開発したくなります。しかし、秘術の陰には基礎テクニックがあるのです。あくまでも、秘術はその延長線上にあることを忘れないでください。

本書にある基礎テクニックは、まさにマシン語のための『秘伝の書』となるべき空気のような存在です。あるいは、「なんだ、こんなもの!!」と思うかもしれません。基礎とはそんなものです。しかし、つまらないことを当り前に使えるか使えないかで、プログラムは大きく変わってくるのです。本書の内容がつまらなく思えなくなった時、オリジナルの秘術的マシン語プログラムが完成する時といえるでしょう……。

なお、本書はZ80をCPUとするコンピュータ用のマシン語プログラムを、アセンブラ上で開発することを前提に書かれています。したがって、プログラムやテクニックを実用化するためには、アセンブラとそれを使用するための基本知識が必要です。

ニモニックやアセンブラの必要性がわからない場合は、まずマシン語の基礎を教えてくれるような本を読んでください。本書では、マシン語を覚えるための基礎は学ぶことができません。あくまでも、「マシン語テクニックの基礎をマシン語を使って学ぶ」ということが目的です。

また、お手持ちのアセンブラによっては、二進数が使用できないとか色々な

制限があることがあります。そういった入門用のアセンブラをお使いの方は、できるだけ早いうちに新しいものを手に入れることを考えたほうがいいでしょう。少なくとも、本書のプログラム程度は問題なくアセンブルできるようなものでないと、マシン語で秘術を凝らすには荷が重いかもしれません。まずは、アセンブラを空気のような存在にすること、それがマシン語の最初の基礎テクニックです。

　では、ごゆっくりと秘伝の世界を楽しんでください……。

二の章

秘伝の問答八十

問 その1 マイナスの数値

はじめまして。ボクはこれまでゲームばかりやっていたので、学校では「動くパソコンソフト」と言われています。でも、実は秘かにゲームから脱皮しようと思い、マシン語を勉強し始めています。マシン語といっても、やっとニモニックの意味とアセンブラの必要性がわかりかけてきた程度です。

ところでボクの読んだ本によると、一般にアセンブラのソースリスト上では十進数表記も可能と書いてあります。といっても、ニモニックで用意された命令で扱える数値は1バイト(0〜FF_H)か2バイト(0〜$FFFF_H$)ですから、十進数でいえば0〜255まで(1バイトの場合)か0〜65535まで(2バイトの場合)となるはずです。でも、でもでも、ですよ。ある時、雑誌に……

```
XPOS:   DB     -10
```

となっているソースリストがあったのです。ハッキリ言って、これがどういう数値を表すのか意味がわかりません。もちろん、十進数ならわかります。いったい、十六進数でマイナス(−)とはいくつのことですか。ちなみに、ボクはまだ本物のアセンブラは持っていません。早くアセンブラがほしい……。

初心者マー君（埼玉）

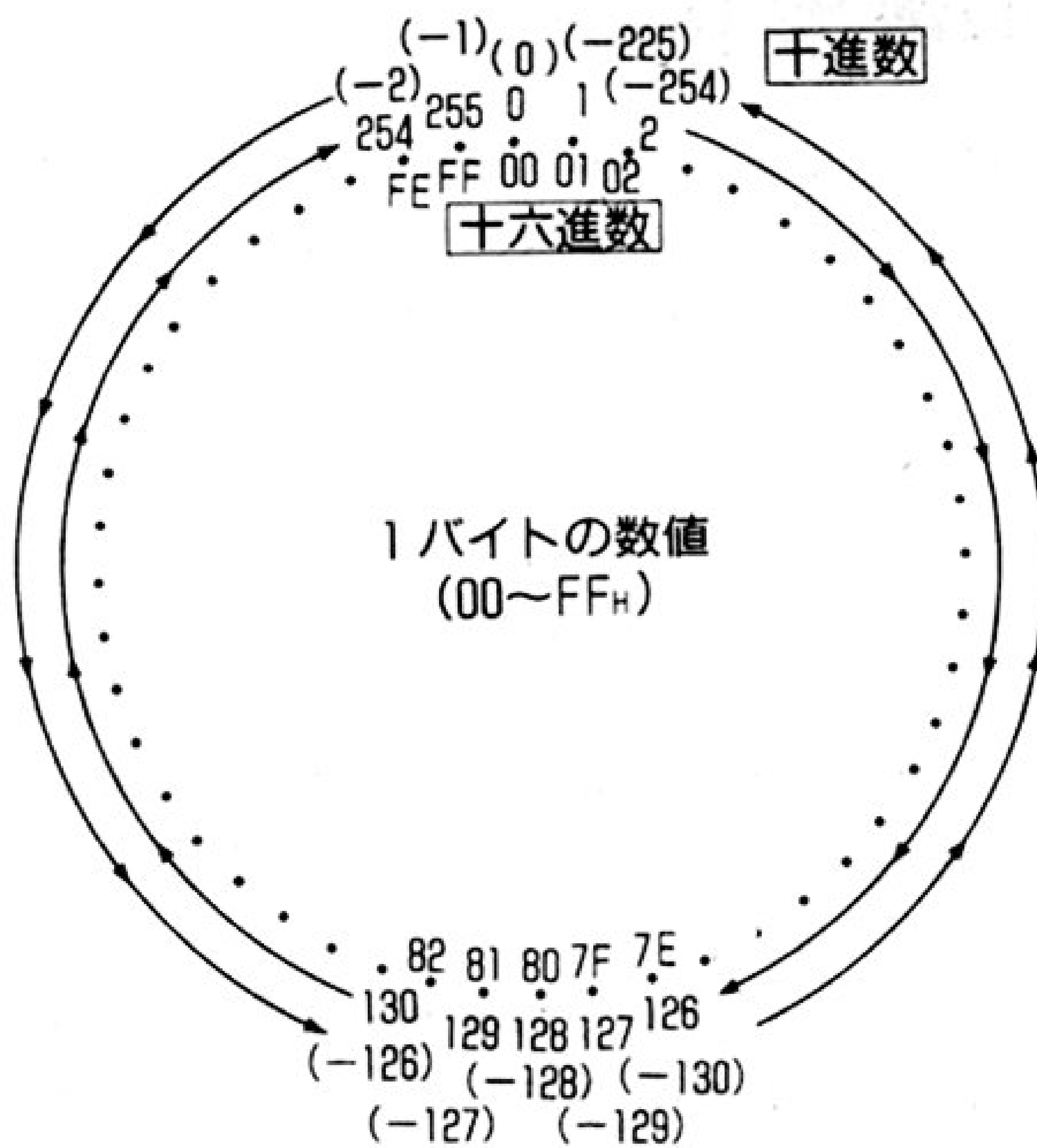
答

初心者からの質問というのは未知数の魅力にあふれていて、内容にかかわらずとても新鮮に感じます。ついつい自分の過去を思いだしたりします。

マシン語が上達するひとつのカギは、自分の持っているアセンブラの能力をどこまで発揮できるかにあります。とはいえ、最初から高級なアセンブラを持てばいいというものでもありません。あまりアセンブラが高級だと、マシン語を覚える前にアセンブラのマニュアルに圧倒されてしまうからです。だから、最初は操作が簡単で安価な入門用アセンブラを手に入れ、自分の技術レベルに合わせてグレードアップするほうがいいでしょう。

ということで、初心者マー君の疑問に一気に答えてしまいましょう。なーに、答は簡単です。マシン語で扱う数値は、使う人の気分次第でプラスにでもマイナスにでもなるのですから。つまり、ループ(FF_Hの次は0)している数値を前向き（プラス方向）に数えるか、後ろ向き（マイナス方向）に数えるかの違

いがあるだけなのです。



だから、−10 は十六進数でいえば F6Hというわけです。もし、ここでの命令が「DB」でなく「DW」であれば、−10 はFFF6Hとなります。でも、でもでも、ですよ。これで若葉マークが取れたなんて思ってはいけません。なぜなら、演算の結果、サインフラグを見て条件分岐をさせる場合は、コンピュータはデータの最上位ビットを見て＋／−を判定するからです。つまり、数値としてはすべてをプラスと考えようがマイナスと考えようが自由ですが、コンピュータにその判断をさせる場合は、次のようにキチンと区別をしなければならないということです。

1バイトの場合：	００H〜７FH	＝ 0〜127	(プラスの数値)
	FFH〜８０H	＝−1〜−128	(マイナスの数値)
2バイトの場合：	００００H〜７FFFH	＝ 0〜32767	(プラスの数値)
	FFFFH〜８０００H	＝−1〜−32768	(マイナスの数値)

マイナスという感覚で数値を扱う時は、ここまで理解した上で使用しないと思わぬバグに陥ることがあります。そして、こういう一見つまらなさそうな理由によるバグが、実は最も恐ろしい落し穴なのです。しかし、マイナスが自由に使えるようになると、プログラムが急に進歩するのも事実です。初心者マー君がプログラ・マー君になる日も近いことでしょう。

なお、雑誌などに発表される簡易的なアセンブラでは、マイナスが使用できないものもあります。プログラム技術の向上のためには、そのようなアセンブラからはできるだけ早く卒業し、せめてマイナスと二進数くらいは扱えるものを準備したいところです。

問その2　慣用句的論理演算

ボクはマシン語を覚え始めたばかりの小６です。まだ本当に短いプログラムしか組めません。ボクのマシン語の先生は中２の兄ですが、その兄もＲＰＧでいうならレベル５くらいみたいです。ボクはレベル２くらいでしょう。

でも、兄は論理演算について一応知っています。ボクも兄に教わったので、論理演算をするとビットがセットされたり、リセットされたりするということはわかるようになりました。ただ、まだそれがどんな時に必要になるのかはわかりませんけど。

ボクは、いま昔の簡単なゲームを見つけて、それを逆アセンブルしてプログラムの仕組みを覚えようとしています。昔のゲームっていうのは、これが本当に商品だったのかと思うほどレベルが低かったのですね。プロテクトなんて、もちろんナシです。でも、結構論理演算なんかも使っているみたいです。

ところが、よく出てくる論理演算に「XOR A」と「OR A」というのがあります。紙に書いて結果を確認すれば、Ａレジスタの値がいくつになるかはわかります。でも、なんのためにやっているのか、その目的がさっぱりわかりません。昔のゲームといっても、結構むずかしいんですね。どうかよろしくお願いします。

少年アセンブラ（岐阜）

答

こりゃマイッタ!?

小学６年生でマシン語を始めてる……だって!?　それも、論理演算にまでコマを進めてる……だって!?　オドロキ桃の木サンショの木……です!!

しかも、マシン語の勉強法に昔のゲームソフトを解析するなんていうのは、将棋でいうなら最初から最善手を指しているようなベストウェイです。なにしろ、昔のゲームというのはプログラムの内容がほとんど画面に反映されるようになっていますから、解析してプログラム手順を覚えるには最高の教科書なのです。もしかすると、アッという間に先生であるお兄さんのレベルを越えてしまうかもしれません。

……が、そうなるためには今回の質問のようにプログラムの目的をしっかりと理解することが大切です。だから、この本が手元にあるってことは、もはや虎の巻を手に入れたようなものです。

さて、まず論理演算による各ビットの変化についてですが、これはマシン語

の入門書であればどんな本にでも載っていることなので、ここでは省きます。問題は、いかにして論理演算の用途を把握するかということです。

その理由は、論理演算については純粋にビット操作が目的の場合と、フラグを変化させるのが目的の場合があるからです。ビット操作が目的の場合は、そのビットの役割を調べればわかりますが、フラグを変化させることが目的の場合はいくらビットの内容を調べても結論は出てきません。

では、どうすればいいかというと、単純に暗記すればいいのです。おもに使用される慣用句的論理演算は次の3種類ですから、暗記というほど大袈裟なことでもありません。とはいえ、使用頻度はビット操作が目的の論理演算より多いくらいですから、とりあえずは深く考えずに覚えてしまいましょう。

XOR　A：演算の結果が常にゼロになりますから、「LD A,0」の代わりによく用いられます。ただし、演算によるフラグ変化がありますから、そのことを考慮した上で使い分けなければなりません。

OR　A：論理演算によるフラグ変化を利用し、「CP 0」の代用として、あるいはキャリーフラグのリセット用としても使われます。

AND　A：「OR A」とまったく同様の目的で使用されます。

これらは単なる慣用句ですから、論理演算の機能を本当に利用したとはいえないのですが、プログラムがちょっぴりオシャレしたような気分になれるでしょう。しかし、本当にカッコイイのは論理演算を2つ以上組み合わせて目的のデータを作り出す、というような使い方をしたプログラムなのです（例えばXORをとってからANDをとる……等）。

その時には、マシン語のレベルも10を超えているでしょう。でも、小6でそこまでカッコイイと、ハッキリいって大人はつらい……。

問 その3　B=0の調べ方

私は現在イギリスはブライトンに住んでいます。本当の目的は一人旅ですが、最初の8週間は英会話のスクールに通い、まずは外国生活に慣れるつもりです。いま、ちょうど半分が過ぎたところですが、夏なのにカラッとした気候のブライトンにいると、このまま住み着きたくなるほどです。

でも、日本では大学5年生に籍を置いたままですから、スクールの後は予定通り旅をして日本へ帰ります。帰国予定は12月です。卒業のほうは、具体的には秘密ですがちゃんと手を打ってありますから大丈夫でしょう。

ところで、どこにいても気になるのは覚えかけたマシン語のことです。だから、英語の辞書とマシン語の本はいつもカバンに入っています。その本には……、

『Aレジスタがゼロかどうかを判定するには、「CP　0」より「OR　A」がいい』

と、書いてあります。理由はプログラムに要するメモリが少なくて済むからだそうですが、同じようなことを他のレジスタでもやってみたいのです。うまい方法があったら教えてください。手元にマシンのない私の答は、次のようなものです。

（例）Bレジスタがゼロかどうかを調べる場合

```
LD    A,B
OR    A
```

W大商学部の星（ブライトン）

答

ム、この男性はどこかで見かけたことがある……。それも、つい最近などというものではなく、なぜか生まれた時から現在まで、そのすべてを知っているような気がしてならないのだが……それが誰なのかはわからない……ウ～ム。

マシン語を始めたばかりは、プログラムの長さとか速度などまるで気にならないものですが、やがてそれに目覚める時が来るのです。その時こそ、マシン語が新たなるパズルゲームに発展する日なのです。あなたにも、ついにその日がやって来ましたか。

質問の回答をする前に、なぜ「CP　0」より「OR　A」がいいのか考えてみ

ましょう。答は簡単です。「CP　0」では、命令にメモリを2バイト使用し、実行に要する時間は7ステートかかりますが、「OR　A」ならばメモリは1バイト、実行時間は4ステートで済むからです。ただし、すべてのフラグが同じ変化をするわけではありませんから、ソックリ同じとはいえませんが……。そのことを知った上でなら、実用上は同じと見なしても差し支えないでしょう。

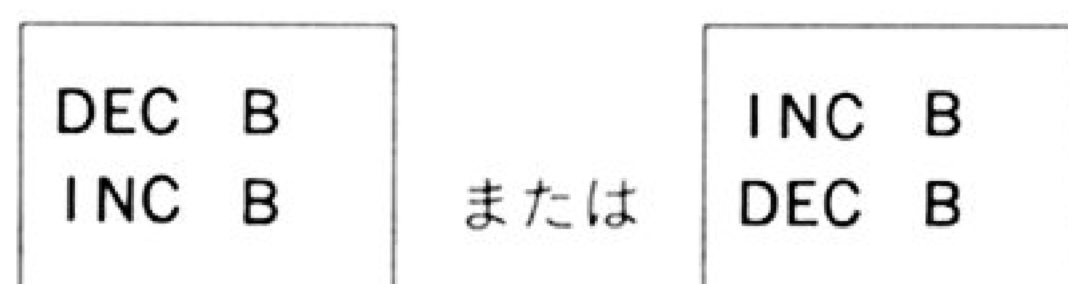
早い＝タイムステート数が少ない

安い＝使用メモリ数が少ない

これが、うまいプログラムというわけです。では、質問への回答です。結論からいうと、Aレジスタが空いているという条件付きでなら、これは正解です。しかし、Aレジスタの値を壊したくないとなると、話は違ってきます。

その場合、この方法ではAレジスタの値を、いったん別のレジスタに退避しなければなりません。空いているレジスタがあればいいのですが、いつもそううまくいくとは限りません。「PUSH AF / POP AF」では、フラグも元の状態にもどってしまうのでダメです。

そこで、こういう場合には次のようにします。



```
DEC  B            INC  B
INC  B    または   DEC  B
```

どちらも命令に要するバイト数、実行ステート数は同じです。足して引いても、引いて足しても、変化するのはフラグだけですから、Aレジスタの内容は壊れないわけです。まさに、パズルではないですか……。

なお、1つの命令に対するバイト数と、ステート数は三の章のZ80ニモニック表を見ればわかります。ステートというのは、その命令に必要な時間の単位ですが、実際の時間はそのマシンの基本周波数（4MHzとか8MHz）によって変わります。

問 その4 RAMの特徴を活かす

ドスコ〜イ!!

朝もハヨからドスコ〜イ。ワシは大横綱千代の富士。

……を目指して、新弟子検査を受けようとしている自称超横綱万代の富士（まよのふじ）っす。身長183センチを目標に、168センチの身体にムチ打って連日ドカ弁を食べているっす。おかげで体重だけは 115Kg もあるっす。ア、いま中三っす。

寝てもさめても相撲だったのに、なんの間違いか誕生祝いにパソコンがほしいと言ってしまったっす。もちろん冗談 100パーセントのつもりだったす。これまでほしいものを一回で買ってもろたことなんてなかったすから。ところがなんと、誕生日の日学校から帰ると、机の上には本物のパソコンが……。

てなわけで、最近では BASIC を通り越してマシン語に夢中っす。ところで、メモリには ROM と RAM があるっすが、ROM だけじゃプログラムは無理ってことは理解してまっす。ワークエリアがなけりゃプログラムは組めないっすもんね。

でも、この前『RAM の特徴を利用したプログラム』があるってことをどこかで耳にしましたっす。これはいったい、なんのこっすか？

どすこいドカ弁（香川）

答

いよいよマシン語も体力勝負の時代に突入したようですね。ドスコ〜イのかけ声と共にマシン語が土俵の外へ突き飛ばされそうです。

さて、プログラムであれば BASIC であれマシン語であれ RAM は絶対に必要ですが、これは基本的には変数保持のためにワークエリアが必要だからです。例えば、カーソルやキャラクタの表示位置を示すワークエリアとして、よく「XPOS」「YPOS」というラベル名を使用しますが、この中身をプログラム中に BC レジスタにロード（C レジスタに X 座標、B レジスタに Y 座標）すると仮定してみましょう。

これを通常のプログラムー①と RAM を利用したプログラムー②とに分けて組んでみたのが次の 2 つです。どこがどう違うか、それぞれの特徴を見極めるつもりでプログラムを見てください。なお、プログラムの右側にある数字は、その命令に必要なバイト数を表しています。

①

```
XPOS:   DB    0              1
YPOS:   DB    0              1
           :
        LD    BC,(XPOS)      4
           :
```

②

```
XPOS:   EQU        WPOS+1   0
YPOS:   EQU        WPOS+2   0
              :
WPOS:   LD         BC,0     3
              :
```

どちらのプログラムも、これ以外の場所で「XPOS」「YPOS」の中身を操作する命令に関してはまったく同じ条件です。ということは、RAM を利用したプログラム－②では通常のプログラム－①より命令に要するメモリ数が 3 バイト少なくて済むということがわかります。

さらに、BC レジスタにデータをロードする命令も、①(20ステート)に対し②は10ステートと半分になっています。したがって、この 3 行分だけに関していえば、メモリ数もタイムステート数も②は①の半分になっているわけです。こうなると、どちらが得かはもう明白です。いかにも、①のプログラムは素人っぽく見えてきますね。

ただし、このようなプログラムが通用するのはプログラム自体が RAM に置かれるということが絶対条件ですから、将来 ROM 化するような場合はキチンとワークエリアを分離した①の書式にしなければなりません。あくまでも「RAM の特徴を利用した」ということを忘れてはならないのです。

しかし、本書のプログラムにおいては、原則的に ROM 化することまでは考えていませんから、この先さらに RAM の特徴を利用したプログラムが現れてきます。その際、特に RAM 専用という断わりを入れませんので、もし ROM 化予定のプログラムに本書のテクニックを応用したい場合は、それなりに RAM エリアを活用するよう、各自で考慮してください。

そもそも、タダで便利なものにはどこかに条件があるというのは世の常なのです。香川のどすこいドカ弁君にパソコンがプレゼントされたのも、きっとその裏には別の願いがあったんではないっすか。身長／体重のバランスからして……。

問その5 HL=0の調べ方

聞いてください。わたしの悩み。マシン語の便利な用法がたくさんでている本を買ったんです。その本には「足して引けば、フラグが変わる(問その3参照)」って書いてありました。早速、わたしは役立てようと思いました。HLレジスタに、ある点数を入れてあるのですが、それがゼロの時には、別のルーチンへジャンプさせようとしたんです。

```
INC   HL
DEC   HL
JP    Z,STUDY
```

これが、その時のプログラムです。でも、HL レジスタの値がゼロになっても、いつも素通りしてしまい、「STUDY 」にジャンプしてくれないんです。あの本にはバグでもあるんでしょうか。

悩める美人のＯＬ（福岡）

答

本書を買ってくれた最初の読者からの質問ですね？？？　それも、美人のＯＬとはウレシイじゃありませんか。

マシン語というのは、蒸気機関車みたいな人間臭さがあるせいか、あまり女性には好かれませんでした。でも、女性がマシン語を操るというだけで、パソコンのイメージが‘ネクラ’から‘ネアカ’に変わりそうです。

さて、このプログラムは一見すると正しいようですが、実はそこに大きな落し穴が潜んでいます。それは、フラグというものに対する過信です。フラグレジスタというレジスタは、思ったより好き嫌いの激しい性格をしています。

ですから、たとえ命令の内容が似ているからといって、フラグレジスタの変化まで同じと思ってはいけません。今回の場合も、「INC HL」「DEC HL」という命令ではフラグはまったく変化してくれないのです。

ということは、ここでやった「INC HL」「DEC HL」という作業は、単なる無駄命令に過ぎなかったわけです。素通りするということは、おそらくこのプ

ログラムの前で、ゼロフラグが立たないような命令を実行していたからでしょう。

では、どうしたらいいかというと、Aレジスタを気楽に拝借すればいいのです。

```
LD      A,H
OR      L
JP      Z,STUDY
```

これは、結果的にはHレジスタとLレジスタのORを取って、ゼロフラグが立つかどうかを確認してジャンプするようにしているわけです。もしAレジスタの値を変化させたくなければ、いったん空いているレジスタに退避しておけばいいですね。かりにDレジスタが空いているとしましょう。

```
LD      D,A
LD      A,H
OR      L
LD      A,D
JP      Z,STUDY
```

簡単なものです、ハイ。では、空いているレジスタがまったくない場合はどうでしょうか。その時は、必殺の書き換えプログラムです。

```
        LD      (MODOS+1),A
        LD      A,H
        OR      L
MODOS  :LD      A,0
        JP      Z,STUDY
```

「(MODOS+1)」とは「LD　A,0」の‘0’の部分に当たります。こうしておけば、レジスタの内容をひとつも壊さずにフラグの変化だけを得られます。

マシン語も女性に好かれるようになると、もっともっと夢のある世界が大きく広がるんですが……。まずは‘悩める美人のＯＬ’さんの悩みを解消したことで、少しは未来が明るくなった気がしませんか。

問その6 無駄命令とは

青春とは爆発だァ!!

オレは高3。あり余るエネルギーを、走ることで燃焼させている。別に陸上部に所属しているわけじゃないから、距離はたったの3 kmさ。だけど、ほとんど毎日のように走っている。正直いって走っている時はメチャ苦しい。顔は苦痛にゆがみ心臓はオーバーヒート寸前……。ここで歩けばどんなに楽かと、いつもそれだけを考えて走っている。

でも、それを我慢して走り終わると、これがまた何とも言えない壮快感。誰もホメてくれなくても、オレはその瞬間が大好きなんだ。無駄なことをしている、なんて言うヤツもいるけどネ。無駄なことができるってのは、ゆとりがある証拠さ……。

ところで、コンピュータにも無駄命令ってのがあるって話だけど、こればかりは自分のことじゃないからよくわからない。ニモニック表を見たら「NOP」という命令があったけど、これが無駄命令のことなんだろうか。

人間無駄蒸気（秋田）

答

いいなァ、無駄なことができるなんて!!

……と言いつつ、私もまだ現役で走っています。でも、それはエネルギーが余っているからではなく健康のためです。もちろんエネルギーは消耗しますが、これは回復力という別のエネルギーを呼び出してくれますからね。決してエネルギーの無駄な消費ではないわけです。

つまり、一見して無駄に見えることが実は無駄なんかではなく、身体にとって重要な役割を果しているということです。コンピュータにおける無駄命令も、本当に無駄なら誰も使わないはずです。と考えると、無駄命令という言葉が存在すること自体、そこには無駄ではない何かがあるはずです。

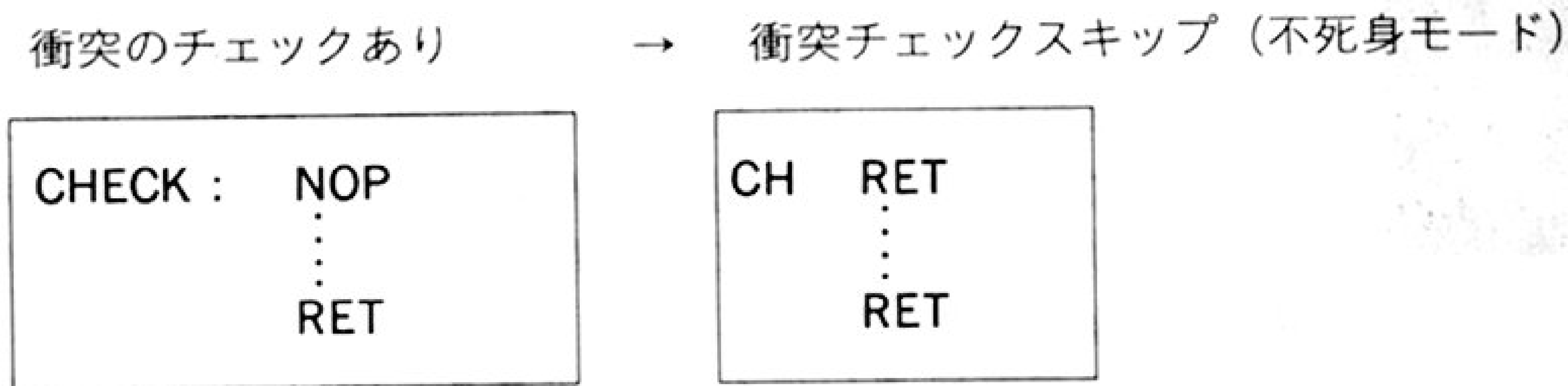
そこで、まずは質問にあった「NOP」という何の役にも立ちそうもない命令を見てみましょう。確かに、この命令はフラグ変化も伴わないし、実行結果を見ても何も残してくれません。でも、これは決して不要な命令なんかではないのです。その証拠に、次のような役に立つことを実行してくれています。

(1) メモリを1バイト確保する。

(2) 4ステートという時間を消費する。

メモリを1バイト確保してくれるということは、必要に応じてそのメモリを

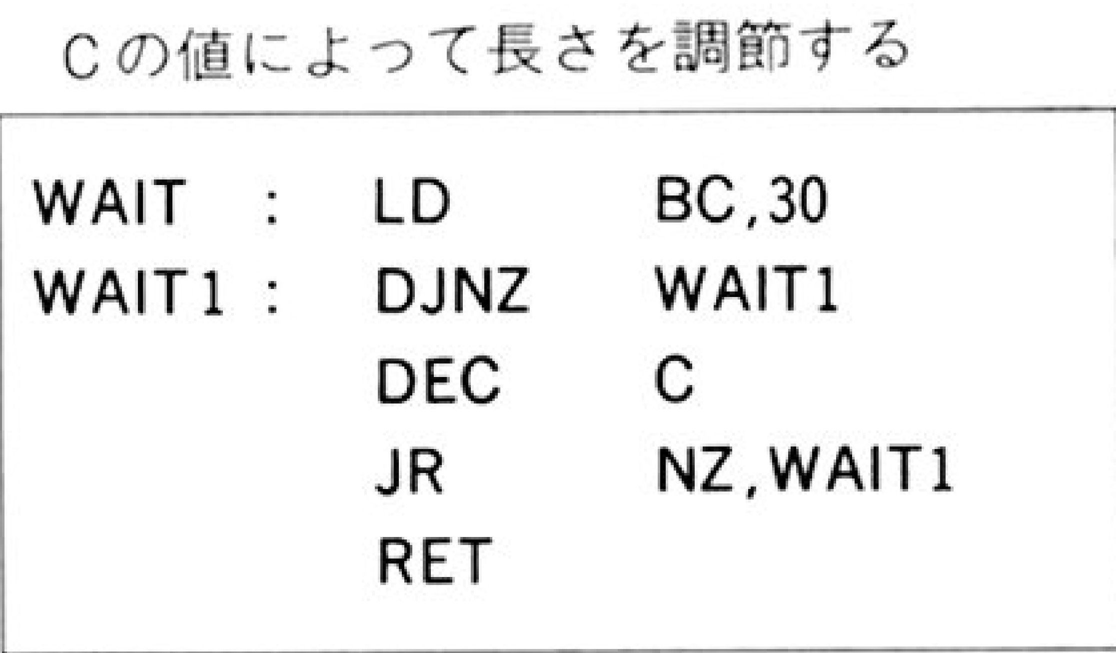
利用できるということです。例えば、衝突をチェックするサブルーチンがあるとすると、その先頭に「RET（=$C9_H$)」を入れれば、簡単に不死身モードができてしまいます。



また、4ステートという時間の消費は、外部機器や特殊デバイスとのヤリトリをする場合に、タイミングを取るための最小ウェイトとして活用されます。

どうですか。これら2つの役割だけでも「NOP」が役に立つ命令であることがわかりますね。もちろん、「NOP」ではなく「LD　A,A」とか「LD　B,B」などといった命令でも代用できますが、「NOP」のいいところはアセンブルされた時に 00_H と数値的にわかりやすい点にあります。

さて、無駄命令といわれているものには、このほかにもタイマーとしてのウェイトルーチンがあります。例えば、カーソルなどはウェイトを入れなければ早すぎて思い通りに動かせませんし、ゲームでもスピードの調節が不可欠です。このような場合、正確には割込みによってキチンとしたウェイトを取らなければなりませんが、簡単なルーチンの場合は次のようにループによってウェイト調整をすることもあります。



このようなプログラムを一般に無駄命令と称しているわけですが、本当に無駄なのはダラダラと不要に長い下手なプログラムというべきです。必要があるから使われているウェイトルーチンは、かわいそうにも俗称だけが無駄命令というわけです。

もっとも、人間の反応がコンピュータ並みに早ければ、大半の俗称無駄命令も省けることになりますが……。そうなると、コンピュータそのものが無駄な存在になってしまいます。

問 その7

「ADD HL, A」を実現

私は、定年退職してからパソコンを始めた者です。息子のパソコンをいじっているうちに、ついマシン語なるものに興味をもちました。ところが、年のせいか、どうも融通がききません。いつも、レジスタが足りなくて「PUSH / POP」ばかりを使ってしまいます。もちろんワークエリアの存在は知っています。しかし、これはワークエリアを使っても解決するような問題ではないと思うのです。

いま、HL レジスタに80を足したいのですが、あいにく DE レジスタも BC レジスタも使用中です。なんとか、「PUSH / POP」地獄から抜けたいのですが……。

```
PUSH    DE
LD      DE,80
ADD     HL,DE
POP     DE
```

やっぱり、これしかナイものでしょうか。私にはこれ以上の案はでませんが、老人でもプログラムの格好は気になるものです。

ボケない老人（栃木）

答

パソコンに年齢なんか関係ありません、という見本のような例です。コンピュータといえば、ついこの間までは大型の電子計算機を意味していたのですから、若い人も年とった人もスタートは一緒です。

そんなことより、新しいことにチャレンジできるなんて、まさに青春まっ盛りではないですか。おまけに、「PUSH / POP」地獄から脱却したいとは、なんて素晴らしいことなのでしょう。プログラムの格好を気にするなんていうのは、プロのプログラマーでもなかなか言えないセリフです。

ところで、このようなケースには実際によく出会います。特に、データの転送などではHL／DEレジスタでデータのアドレスを指定し、BCレジスタをカウンタに使うことが多いですから、もしもHL／DE両レジスタとも途中で＋80するなどという場合は、「PUSH / POP」を使っていては確かに大変です。もちろん、ワークエリアを使って退避することもできますが、それでは余りに

も大袈裟です。そこで、空いているAレジスタを利用してみましょう。

```
         LD      A,L
         ADD     A,80
         JR      NC,NOINC
         INC     H
NOINC :  LD      L,A
```

質問にあった「PUSH / POP」を使用した方法では、使用メモリ数＝6バイト、ステート数＝42となります。一方、こちらの方法では、使用メモリ数＝7バイト、ステート数＝27（INC H を通る時は26）です。

これだけでは、総合して少し優位という程度ですが、この方法によるメリットはレジスタを選ばないという点にあります。というのは、ペアレジスタ同士の計算命令では、常にHLレジスタを介在させなければなりませんから、HLレジスタの値を保存したままDEレジスタを ＋80する時など面倒です。そんな時には、圧倒的にこちらの方法が便利になるわけです。

では、栃木のおじいちゃん、格好のいいマシン語プログラムを目指して、ますます頑張ってください。



問 その8

「ADD HL, 1234H」を実現

与論島をご存じですか。そう、若者の島、ヨロン島です。沖縄が日本に返還されるまでは、日本最南端の島といわれていました。

わたしの家は民宿をしています。ヨロンにはヨロン憲法（献奉）という歓迎の挨拶があって、島に来た人はまず焼酎をなみなみと飲まされるんですよ。わたしの役目は飲むふりをして飲ませることです。だって、次の日は学校だもん。

ここの海岸にはコンペイ糖みたいな形をした星の砂がたくさんあります。有名だから知ってますよね。わたしは、いま「星の砂」というタイトルのゲームをパソコンで作っています。一応マシン語を使っているのですが、困ったことに HL レジスタに加算する命令はあっても、BC レジスタに加算する命令はありません。

```
PUSH    HL
LD      HL,1234H
ADD     HL,BC
LD      B,H
LD      C,L
POP     HL
```

BC レジスタに 1234_H を足す方法は、これが最善の方法ですか。実は、ある本で１バイトの数値をペアレジスタに加算する方法は発見したのですが……。同じような方法がありそうな気がします。気のせいでしょうか。

星降る乙女（与論島）

答

ヨロン島の星降る乙女ちゃんの作品で「星の砂」……。もう目の前にメルヘンの世界が広がってきそうです。ヨロンには潮が引くと沖合いに島のように現れる百合が浜なんていうのもあるし、夢がいっぱいですね。

でも、お酒を飲めない人にとっては、ヨロン憲法は余りにもキビシイ憲法です。その割には、毎日の宴会は楽しかったけど……。だから、質問にピッタリの答を教えてしまいましょう。

おそらく、ペアレジスタに１バイトの数値を加算する方法というのは、問(その7）にある答だと思います。そこで、あのプログラムをもう少しヒネって、

HLレジスタを介さずにペアレジスタへ直接数値の加算（ここではBCレジスタに1234_Hを加算）が行えるようにしてみます。

```
LD      A,C
ADD     A,34H
LD      C,A
LD      A,B
ADC     A,12H     ← ADCとしている点に注意してください
LD      B,A
```

たったこれだけですが、ペアレジスタに直接2バイトの数値が加算できるということになると、用途はかなり広くありそうです。必要に応じて1バイトの数値加算法と使い分けると、速度的な無駄もなくなります。

また、加算するのが数値でなくレジスタであっても同様の書式（数値を加算している部分をレジスタに置き換える）が可能ですから、「ADD　BC,HL」とか「ADD　DE,BC」なんていうのも、いとも簡単に作り出せるわけです。大いに活用してしまいましょう。

ところで、星の砂ってどうしてあんなにきれいな星の形をしているんでしょうか。一説によると、ヨロンのあの満天に散りばめた星の群れから、時々はみ出して落ちてきたという話ですが、本当のことを知っていたら教えてください。星降る乙女さま……。



問 その9 「SUB HL, DE」を実現

コンニチワ。こちらは冬が長い北の国です。寒い日は、家の中でパソコンをするのが最高です。特に、マシン語をプログラムしていると時のたつのを忘れます。今はまだ入門者ですが、そのうちアッと驚くプログラムを組んでみせます。

ところで、いつも不便に思うことがあります。それは、ペアレジスタの減算命令についてです。足し算には「ADD HL,ss」と「ADC HL,ss」という2つの命令があるのに、減算には「SBC HL,ss」という命令しかありません。

だいたい、キャリーフラグ込みの減算なんて使ったことがありません。だから、毎回キャリーフラグをリセットしていますが、どう考えてもこれは不親切な命令だと思いませんか？

ちなみに、キャリーフラグのリセットには「OR　A」とやっています。でも、気分によっては「AND　A」を使ったりします。

夜ふかしヒグマ（北海道）

答

雪がしんしんと降る中をプログラムか……。うらやましい環境だなァ。きっと、プログラムに疲れたら、近くの山にスキーにでも行くんでしょうね。いいプログラムはいい環境のもとで生まれるとよくいいます。タバコの煙と、ほこりにまみれた部屋からは、それなりのプログラムしかできないという気がします。

きっと、近いうちに日本を揺るがすようなプログラムができることでしょう。なんといっても、本人がその気になっているのが強みです。

それにしても、本当に「SUB　HL,ss」という命令があってもよさそうです。あれば便利なことも間違いありません。でも、「SUB　HL,ss」と「SBC　HL,ss」どちらか1つだけを選べと言われれば、私ならやはり「SBC　HL,ss」を選びます。

それは、キャリーフラグをリセットして減算するのは簡単ですが、キャリーフラグ込みの減算を「SUB　HL,ss」で実現するとなると、「OR　A」みたいに1命令では済まないからです。質問では、キャリーフラグ込みの減算をしたことがないとのことですが、きっと近い将来キャリーフラグ込みの減算が必要になってくるはずです。

とはいえ、現時点ではキャリーフラグを毎回リセットしているわけですから、面倒であることも事実です。なんとか工夫してみましょう。

例えば、HLレジスタから 5000_H を減算するとします。まず、まともにプログラムを組んでみます。おそらく、夜ふかしヒグマクンもこのように組むはずです。

```
LD    DE,5000H
OR    A
SBC   HL,DE
```

キャリーフラグをリセットするのに、「OR　A」を知っているなんてスゴイですね。キャリーフラグにはセットする命令（SCF）と反転する命令（CCF）はあっても、リセットする命令はありません。そのため、「OR　A」とか「AND　A」という命令でキャリーフラグをリセットするのが一般的です。しかし、わずか1バイトの最短命令とはいえ、毎回となるとイヤになってきます。そこで、コロンブスの卵です。

```
LD     DE,−5000H
ADD    HL,DE
```

これで、HLレジスタの値は先ほどと同じ結果になります。減算に要するメモリ数／ステート数も、3バイト（19ステート）から1バイト（11ステート）へと減少しました。アイデアの勝利です。

……が、計算の結果、ゼロフラグを見て何かをさせようという場合はこのままではいけません。というのは、「ADD　HL,ss」という命令ではゼロフラグが変化してくれないからです。

そこのところを理解した上で、「SBC　HL,ss」にするかこちらの方法にするか、うまく使い分けることが大切です。問（その5）にもあったように、ペアレジスタの加減算命令でフラグ判定をする場合は、そのあたりを頭に入れておかないと迷宮入りとなる可能性があります。こわいですね……。

問 その10

JRとJPの違い

私は鉄道マニアです。国鉄（アッ……いまはＪＲでしたネ）の全路線を踏破するのが目標ですが、お金と時間のかかることなのでいつ達成できるか、自分にもさっぱりわかりません。

そこで、パソコンを使って一足お先に画面上で全国踏破しようと思うのですが、今度はプログラムで頭を使いデータ入力で時間を使い、やっぱりいつ達成できるのかわかりません。それでも、こちらのほうはお金がかからないだけ助かってはいますけど……。

実は、最初はプログラムをBASICで組んでいたのですが、あまりに遅いためマシン語でやろうと決心したのです。ただ、マシン語でまともなプログラムを組むのはこれが初めてなので、マシン語といっても幼稚なものです。そこで、恥ずかしいのですが未熟者からの質問と思って教えてください。よく雑誌などで……

> JR命令よりもJP命令のほうがメモリは1バイト多く使用するが、実行に要するステート数が少ないので、JP命令を使用して高速化を図っています。

……と、書いてある投稿記事があります。確かにニモニック表ではJR命令は12ステートで、JP命令は10ステートです。でも、実際にやってみると、効果はナイも同然です。これは一体どういうことでしょう。

ちなみに、現在のプログラムは乗車ルート決めるという段階で、一応思考ルーチンとなっています。だから、本当に高速化されているならば、思考時間の短縮となって表れるはずなんですが……。

ＪＰ不信のＪＲマニア（神奈川）

答

この答は簡単ですね。JP命令だって人の子、いやマシンの子です。ＪＲマニアのあなたがＪＲを嫌いになればいいのです。そうすれば、JP命令はよろこんで超高速に命令を実行するように……というのはウソです。

いきなりウソをついたのは、この質問が決して幼稚ではないからです。それは、JR命令よりJP命令が絶対に早いと単純に思っているJP命令高速信奉者が圧倒的に多いという事実からもわかります。確かに、そのこと自体は間違ってはいません。絶対ジャンプ命令においては、JP命令のほうが2ステート早いことは事実なのですから。

そこで、まずはこのステートというのは具体的にどの程度の時間をいうのか

計算してみましょう。ただし、割込みやDMAなどによるウェイトは考慮に入れません。また、基本周波数は4 MHzとしています。

1ステート＝1÷4÷1000÷1000 秒
　　　　　＝0.00000025 秒

これが1ステートの実際の長さです。つまり、JR命令をJP命令に置き換えると、それだけで 0.0000005秒高速になるというわけです。

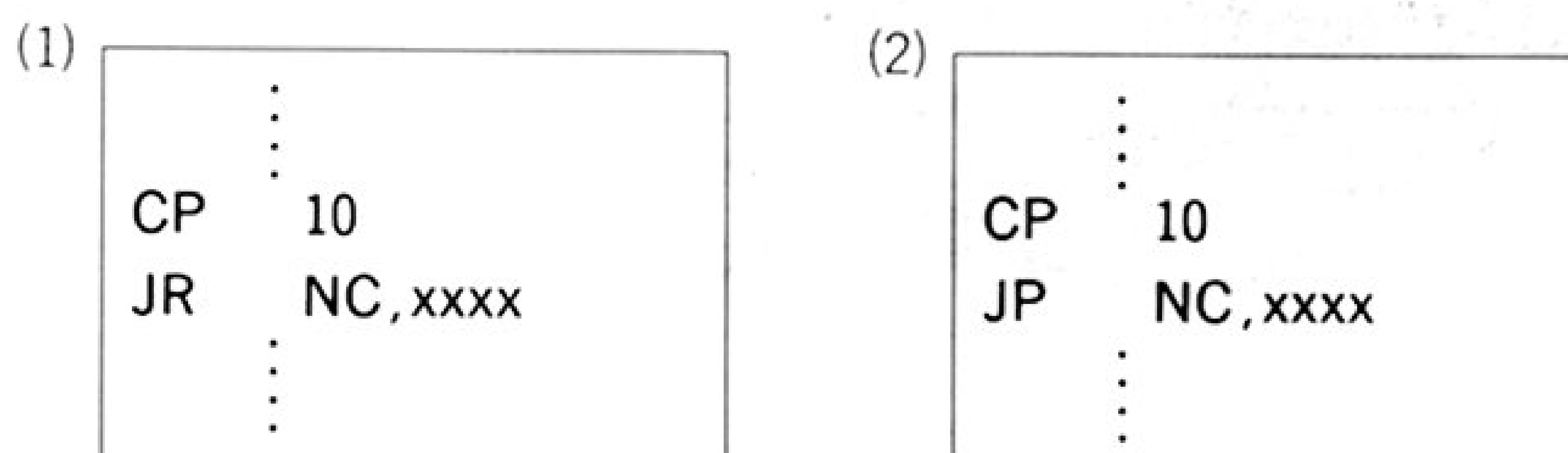
……が、それを確認できるほど人間の頭脳は高速でしょうか。さらに、この方法だけで1秒短縮しようとすれば、JP命令を 200万回実行しなければなりません。つまり、よほどのプログラムでない限り目に見える変化はないということです。

しかし、だからJP命令による高速化が不要ということではありません。例えば、ループの中とか何度も使用するサブルーチンなどでは、高速化への1つの手段として実行する価値は大いにあります。特に、リアルタイム・ゲームでは1ループ（全キャラクタの1コマ移動）の時間が短いので、こういった小さな高速化を集めることはプログラミングの基本となっています。

また、場合によってはループにしていたものを、その数だけ同じ命令を連記してカウントに要する時間を短縮するとか、サブルーチン化をやめてCALL／RETに要する時間を省くというテクニックも有効でしょう。

しかし、本当に重要なのは命令そのものをいじることより、アルゴリズムに対する工夫です。特に思考ルーチンでは、アルゴリズムを変えることにより思考時間が半分になったりすることもあります。もっと視野を広げて、プログラム全体を見つめることが大切なのです。

ということで、最後にJR命令にも花を持たせましょう。それは、同じジャンプでも条件分岐命令による場合です。2つのプログラムを見比べてください。



(1)

```
        :
CP      10
JR      NC,xxxx
        :
```

(2)

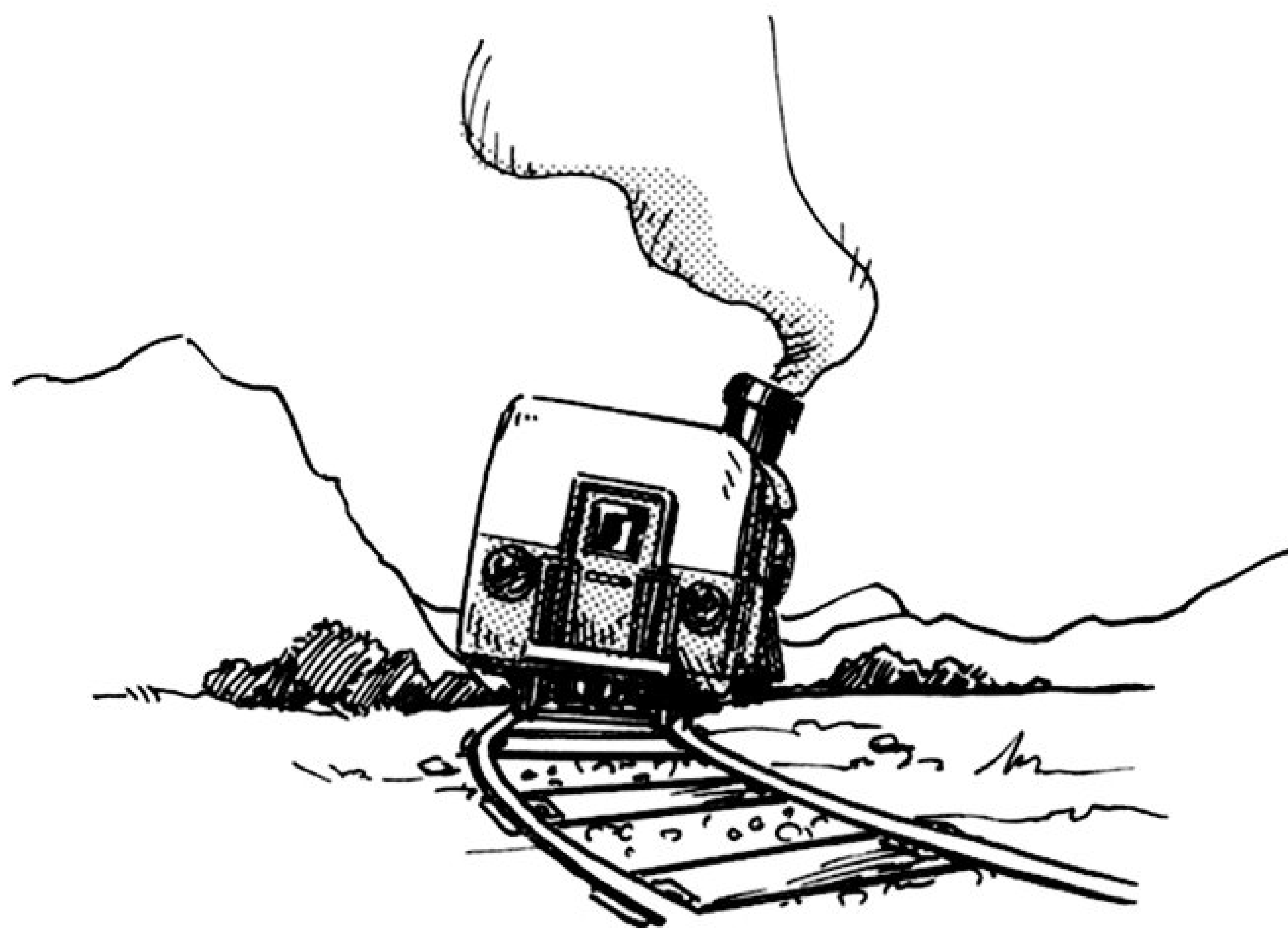
```
        :
CP      10
JP      NC,xxxx
        :
```

JP命令高速信奉者は、ためらわずに (2)のようにしますが、条件分岐というの

は当然のことながら分岐しない場合があるわけです。その際、JR命令なら7ステートで通過してくれますが、JP命令では分岐するしないにかかわらず10ステートを消費してしまいます。

したがって、このようなケースでは分岐する割合が多ければ(2)、少なければ(1)というように、その内容によってプログラムを使い分けることが、真の高速化ルーチン信奉者の条件というわけです。

JR命令は単に使用メモリ数が少ないというだけでなく、リロケータブルなプログラムに使用できるという特技もあるのです。決してJP命令より格が下だなんて思わないでください。新幹線だってＪＲで動いているんですから……？？？



問 その11　＋／－を交互に

オス。おいどんは、薩摩隼人でごわす。時代の波に乗り遅れぬよう、おいどんもパソコンを買ったでごわす。BASIC は、ハッキリいって遅くてイライラする。やはり、時代はマシン語でごわすな。

おいどんは、未来を的確に見つめるため、いまあるプログラムを作っておるのだが、ある事情でコールするたびに ＋50と －50という値を交互に返してくれるようなサブルーチンが必要なのでごわす。現在は次のようにしておるが、どうもこのプログラムはスッキリしないのでごわす……。

```
DAT50 :  LD      A,50
         CP      50
         JR      Z,DAT51
         LD      A,50
         JR      DAT52
DAT51 :  LD      A,−50
DAT52 :  LD      (DAT50+1),A
         RET
```

ぜひ、的確な評価をしてほしいでごわす。お願いでごわす。

マシン西郷（鹿児島）

答

これはこれは、時代を超越したような質問でごわすな。こちらまで影響を受けてしまいそうでごわす。

このプログラムは、確かに見るからにスッキリしませんね。プログラムを書き換えながら使うなど、苦労の跡はうかがえるのですが、苦労がむくわれてないようです。でも、プログラムがスッキリしないと感じている点に、マシン語の夜明けを感じます。

マシン語プログラムで、スッキリしないとか不格好だと感じた時は、命令の一覧表（巻末のインストラクション表）を見直すと、いい知恵が浮かぶことがあります。それは、マシン語を覚えたてのころは、使い慣れたニモニック以外使わない（使えない）という傾向が強いからです。

この西郷クンも、便利な命令をまだ見落としているのです。こんな時には、

「NEG」という最適の命令があるんですよ。

```
DAT50 :  LD      A,50
         NEG
         LD      (DAT50+1),A
         RET
```

「NEG」とは、NEGATE（否定する、正負を反転するという意味）の略ですが、この命令を使えば一発で正なら負へ、負なら正へと変換してくれるのです。もちろん、ここでの正とか負というのは、問（その1）にあったように、あくまでもプログラムを組む人の感覚であることを忘れないでください。一応サインフラグ上では、ビット7の値で＋／－を判断するようになっていますが……。

時計を読むのに、何時何分過ぎと読むか、何分前と読むか、それは読む人の勝手というようなものです。では、同じような感覚で 0と何かを交互に返すルーチン、と問題を変えたらどうでしょうか。例えば、0 と50を交互に返してくれるようなルーチンがほしい場合です。もちろん、質問にあるようなプログラムにもどってしまうようでは、維新の夜明けは遠くなるばかりです。

```
DAT50 :  LD      A,0
         XOR     50
         LD      (DAT50+1),A
         RET
```

今度は論理演算 XOR です。XOR は二度繰り返すと値が元にもどるという特徴を利用するわけです。では、初期値を 0ではなく別の値にしたらどう変化するか、XOR に対する理解を深めるためにも自分で確認してみてください。

よくわからないという人は、キチンと二進数にして演算すればいいでしょう。ナニ、そんな簡単なこと知っていたでごわすか!!　こりゃ、失礼したでごわす。

問その12 LD命令を減らす

わては大阪の生まれだす。手紙も電話も大阪弁以外ではしませんのや。もちろん、英語だって大阪なまりだす。それが、浪花のド根性と思うとんのや。

そ、そのわてが、なんとマシン語に凝ってしもたのや……。くやしいけど、わてのパソコンかて大阪弁は理解してくれへん。これはホンマに残念なことやで。マ、それはいいとして、わてのマシン語というと、ロード(LD)命令の連続なんや。

やっと、ワークエリアが自由に使えるようになったんやが、その中身を操作するたびにロード、ロードや。例えば、こんな具合いや。

```
LD    A,(OKANE)
SUB   2
LD    (OKANE),A
LD    A,(YAKSO)
INC   A
LD    (YAKSO),A
```

「OKANE」とか「YAKSO」というのがワークエリアなんやけど、アドレスでいうたらお隣りどうしやで。もっと、わての大阪弁みたいに流ちょうにいかんのやろか。ア、プログラムの内容はお金を使って薬草を買うた、という意味や……。

浪花ぼてじゃこ物語（大阪）

答

英語いうたかて世界中どこでも通用するわけではあらへん。わてもイタリアを放浪しよった時は、大阪弁使いよったで……デタラメの……。

外国人というと、みな英語を話すと思ってしまうのが島国日本人の悲しさ。苦労して下手な英語を使っても、通じない国はたくさんあります。どうせ通じないのなら、日本語でも同じこと。真剣に話すと、結構相手もわかってくれるようです。

その日本語でも流ちょうに話すのは難しいのに、マシン語を流ちょうに使いたいとは驚きです。凡人には、なかなか言えないことです。

では、さっそく流ちょうでないプログラムを見てみましょう。流ちょうに見えない原因は、アドレスをすべてダイレクトに指定しているからです。忘れな

いでください、ペアレジスタというアドレスを示せるレジスタがあることを。

かりに、「OKANE」と「YAKSO」というアドレスが、番地の若い順に並んでいるとしましょう。一方のアドレスを指定すれば、隣りのアドレスは簡単にわかります。流ちょうな日本語ができなくたって、隣りの家をいちいち住所で言う人はいませんね。どこか一軒を指定すれば、あとは一軒先とか手前というように相対的に表現するのが普通です。

```
LD     HL,OKANE
DEC    (HL)
DEC    (HL)
INC    HL
INC    (HL)
```

これで、プログラムに要するメモリ数は半分以下になりました。ステート数も2割ほど減少しています。気になったロード命令も1つになりました。……が、ロード命令の多少とプログラムの良否については、まったく関係ありません。

そのことより、メモリ中のデータを操作する場合、できるだけアドレスをレジスタで指定するように心がけることが大切でしょう。ほとんどの場合、それだけでメモリ数／ステート数の減少につながります。

ただし、「DEC　(HL)」や「INC　(HL)」という命令は、メモリは1バイトしか使いませんが、時間は11ステートもかかります。加算する数が3以上の場合は、「LD　A,(HL)」としてAレジスタで計算してからメモリにストアしたほうが得策です。

また、関連するワークエリアのアドレスが前後に散らばっていたりする場合は、インデックスレジスタ（IX／IY）を使用するとプログラムはスッキリします。しかし、インデックスレジスタに関する命令は、メモリ数／ステート数ともに大量消費の王様ですから、高速化追求のプログラムでは使用する価値があるかどうか、よく考えてから使うことがポイントです。

もっともマシン語プログラムは超高速で実行されますし、最終的にウェイトをかけて時間調整をすることも少なくありません。したがって、そのあたりを考慮して命令を使い分けることが大切なのです。

どないだ。これで、マシン語も大阪弁に一歩近づいたでっしゃろ!!

問 その13 ビット転送

ニッポンのみなさ～ん。お元気ですか……。

ワタシはおじいさんとおばあさんが日本人の日系三世です。英語名はドナルド、日本名は拓です。だから、ドナルド・タクというわけです。もちろん、通称はドナルド・ダックです。

でも、初めて日本へ行った時‘ドナルド’って呼ばれても誰のことかサッパリわかりませんでした。それに、日本語でいう『マクドナルド』というお店、これも行ってみるまで何のお店かわかりませんでした。ヤッパリ、ワタシはアメリカ人です。

ところが、うれしいことに日本人もマシン語を使うというじゃありませんか。それならというわけで、ワタシもマシン語を始めました。だから、まだ入門者です。そこで、ちょっと教えてください。

Aレジスタのビット5だけを、あるメモリにそのまま移したいのです。ビット5以外は変更したくありません。いまは次のようにしています。

```
          LD     HL,MEMRY
          BIT    5,A
          JR     Z,RSET5
          SET    5,(HL)
          JR     XSET5
RSET5 :   RES    5,(HL)
XSET5 :           ⋮
```

ヤッパリ、こんなもんでしょうか。

Donald Tack（ロサンゼルス）

答

実は、私も憧れのディズニーランド（東京ではなくロスのほうです）へ行く前日、現地の人に明日はドナルドダックに会いたいと言ったのですが、いくら「ドナルド」「ドナルード」「ドーナルド」と言ってもわかってもらえなかったことがあります。

あれを『ダーナーダック』って発音するなんて……。『マクダーナー』というお店も日本にはないですからね。カタカナの英語がこんなにデタラメだとは、その時まで知りませんでした。ちなみに、私が話した現地の人というのは、一応日本語も話せる日系三世の人だったんですが……。

こうなると、マシン語を国際語にするしか方法はなさそうです。質問にも快

く答えてしまいましょう。

まず、質問にあるプログラムは、少し手直すだけで無駄なジャンプ命令が省けてスッキリさせることができます。

```
           LD      HL,MEMRY
           RES     5,(HL)
           BIT     5,A
           JR      Z,RSET5
           SET     5,(HL)
RSET5 :            :
```

これだけで十分と言えないこともありませんが、この程度のことでジャンプ命令を使うのはカッコ悪いと感じる人もいるでしょう。そこで、Aレジスタの値はこわれてもいいという条件で、次のようにプログラムを組むのも一考です。

```
           LD      HL,MEMRY
           RES     5,(HL)
           AND     00100000B
           OR      (HL)
           LD      (HL),A
                :
```

「なんだ、これなら前のほうが早いぞ!!」なんて思いませんでしたか。確かに、この例に関してはそう言われても仕方ありません。では、もしもビット５とビット３の値を移したいとなったらどうでしょうか。

最初のやり方では、プログラムが単純に倍になってしまいます。それに対して、二番目の方法ならRES命令を１行追加するだけで済みます。もちろん、AND命令のオペランドも直しますが、これはメモリ数や速度に影響を与えるものではありません。

つまり、両方を知っていればプログラムに合わせて効率のいい方法を選べるということなのです。やはり、何事においても基本は大切ですね。

英語の基本はＡＢＣ……、決してカタカナではなかったのです。

問その14 サブルーチンへのジャンプ

ボクの友人にパソコンを小学生の時から持っている上、すでにマシン語までマスターしてしまった者がいます。ボクは、1年前やっと高校入学と同時に買ってもらいました。そして、なんとかBASICは彼と同レベルになりましたが、マシン語ではまだまだその友人にかないません。そこで彼に内緒で質問します。

```
      :
CALL    xxxx
RET
```

マシン語プログラムには、こういうケースがよくでてきます。ボクも最初はこのままでしたが、ある時コールしてからリターンするなら、「JP　xxxx」とすればいいのではと思ったのです。どうせ、コール先のRET命令で元のルーチンへ戻れるからです。

以来、このような場合には「JP　xxxx」としてきましたが、それで問題が起きたことは一度もありません。でも、ボクの友人にその話をしたら、いずれダメな時も出てくると言います。理由を聞いてもハッキリ教えてくれません。

いったい、どういう時にダメなんでしょうか。ソッと教えてください。ボクは、今でも大丈夫だと思っています。

ドラQ（新潟）

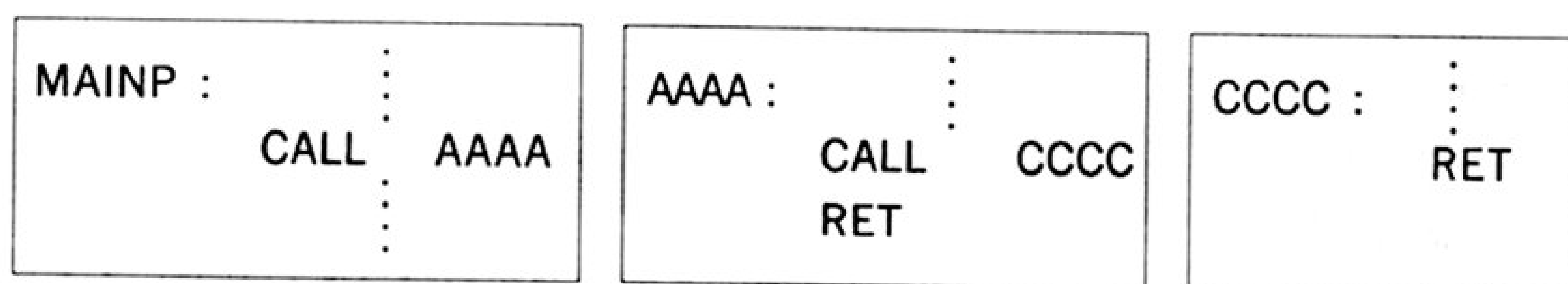
答

友人に自慢されている様子がよくわかります。似たような経験は誰しも一度はあるものです。でも、たったの1年でこんな鋭い疑問がわくのも、その友人のおかげと思えばいいじゃないですか。

この問題は、それほど奥が深いのです。が、通常の場合は質問にあるような手法で困ることはありません。それどころか、わずかですがメモリの節約、スピードのアップ（あくまでも計算上です）になっています。

では、どんな時に困るかというと、それはスタックを操作しながらプログラムが走っている場合です。例えば、メインプログラムからサブルーチン「AAAA」がコールされ、サブルーチン「AAAA」の最後でサブルーチン「CCCC」がコールされているとします。

確かに、ここでサブルーチン「AAAA」からサブルーチン「CCCC」へジャ



ンプすれば、サブルーチン「CCCC」のRET命令でメインプログラムへともどります。しかし、サブルーチン「CCCC」のプログラムが、条件によってはメインプログラムへ直接リターンするように設計されていたらどうでしょうか。

つまり、サブルーチン「CCCC」が「MAINP」から数えて常に二段目にコールされるようになっていて、条件によってはスタックを操作（スタックポインタを+2）して「AAAA」へもどらずにダイレクトに「MAINP」へリターンするような場合です。もし「JP　CCCC」でこのようなケースに出会うと、スタックが狂ってしまい「MAINP」を通り越してどこかへ暴走してしまうことになります。

もちろん、プログラムを作った本人がこのような作り方をしなければ問題はありませんが、共同で大きなプログラムを開発したり、他人のプログラムを借用したり、あるいはシステムの内部ルーチンをコールする場合など、こういった危険性はアチコチに潜んでいるといえます。

具体的な例をあげてみましょう。画面上にピョンピョン跳ねている小さなボールが1つあるとします。ボールは画面に散らばっている釘に触れると破裂してしまいます。わかりやすくするため、設定はたったこれだけです。

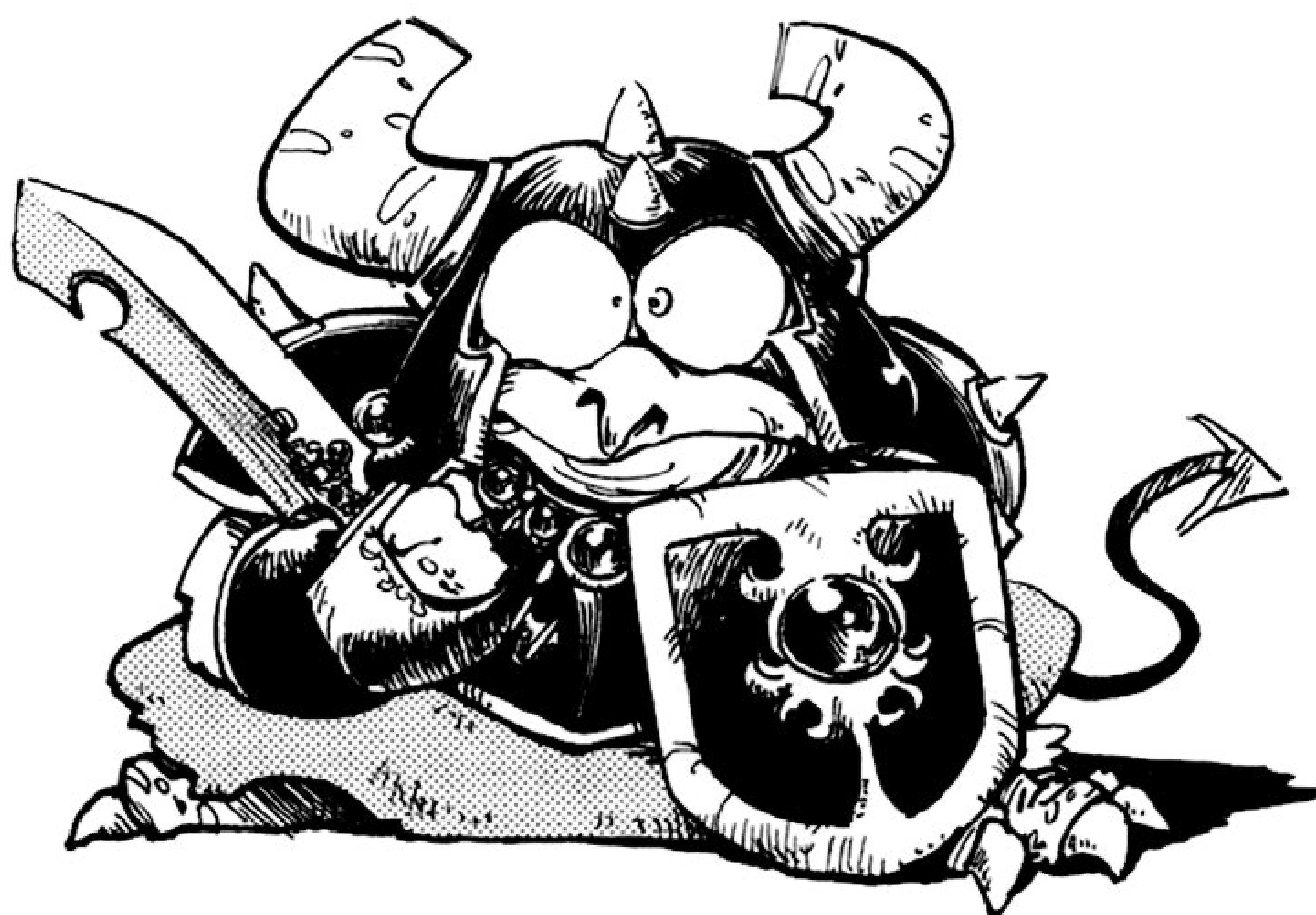
サブルーチン「AAAA」はボールとすべての釘との衝突を管理するルーチンで、ボールの動きを管理している「MAINP」からコールされています。「AAAA」では、HLレジスタにボールの座標、BCレジスタに釘の座標を入れ、次々と座標チェックと破裂を実行するサブルーチン「CCCC」をコールします。

サブルーチン「CCCC」では、ボールと釘との衝突チェックを行いますが、ひとつでも釘とボールが衝突していたら、その時点でボールは破裂し残りの釘との衝突チェックは不要になります。つまり、サブルーチン「CCCC」からサブルーチン「AAAA」へもどるのではなく、直接「MAINP」へもどるケースが生じるわけです。これを実現しているのが、破裂処理のあとにあるダミーのスタック操作です。このスタック操作が成立する前提として、スタックの条件（レベル）を統一するようなプログラムが要求されているわけです。

```
AAAA:  LD    HL,(BALXY)
       LD    BC,(KUGZ1)
       CALL  CCCC
       LD    BC,(KUGZ2)
       CALL  CCCC
       LD    BC,(KUGZ3)
       CALL  CCCC
               :
       RET
```

```
CCCC:  LD    A,H
       CP    B
       RE    NZ        ← AAAA へもどる
       LD    A,L
       CP    C
       RE    NZ        ← AAAA へもどる
               :       ←破裂のプログラム
       POP   AF        ←ダミー
       RET             ← MAINP へもどる
```

面倒そうですが、いずれプログラムに慣れてくると、このダミーを2つ以上連続させたり、常にスタックを考えながらプログラムを組むようになるのです。新潟のドラQ君がそうなるのも時間の問題です。もう友人にも自慢されなくてすむでしょう。



問その15

命令の書き換え

我輩は猫である。名前はまだない。猫に小判という諺はあるが、猫にコンピュータという諺はない。だから、猫がコンピュータに興味を持ってもおかしくはない。

……という書き出しで始まる『猫とコンピュータ』という小説を知っていますか。おそらく、まだ誰も知らないはずです。というのは、まだ完成していないからです。私がその作者ですから、それは間違いのない事実です。

実は、この猫は超能力を持った猫で、プログラマーである飼い主に、プログラムのバグや欠点を教えるというのが話の大筋です。しかし、作者である私がプログラムの初心者なので、欠点を直すというのが大変なことなのです。それが、小説が進行しない最大の原因となっています。

いま、小説の中のプログラマー氏が次のような２つのプログラムを組みました。その２つはほとんど同じ内容のルーチンで、まん中の一部が違っているだけです。どのような助言を猫が与えるべきでしょうか。

```
ADDA1 :       :
              :
         INC    A
              :
              :
         RET
```

```
SUBA1 :       :
              :
         DEC    A
              :
              :
         RET
```

いま考えているのは、前半と後半をサブルーチンとして共通化する方法ですが、そうすると前半の共通ルーチンの途中でリターンしたら困りそうです。

ペンネーム猫目漱石（愛媛）

答

岡目八目ならぬ猫目八目というわけですか。しかし、猫の先生にあたる作者がプログラムをよくわからないとなると、超能力猫からバグ猫に……。下手をすると、化け猫小説となってしまいそうです。

そこで、質問にある修正案通りにプログラムを組んだことにして、それが超能力猫の判断に見合うかどうか、まずチェックしてみましょう。

「HALF1」というのが前半の共通ルーチンで、「HALF2」が後半の共通ルーチンというわけです。一見すると、うまくまとまったような気がしますが、問題は「HALF1」のプログラムの途中でリターンすることがある場合です。

```
ADDA1 :   CALL    HALF1
          INC     A
          JR      HALF2
SUBA1 :   CALL    HALF1
          DEC     A
HALF2 :       :
          RET

HALF1 :       :
          RET
```

つまり「HALF1」の途中でのリターンは、それぞれ「ADDA1」「SUBA1」からのリターンを意味していますから、この場合「HALF1」の中で特別にスタック操作をしてリターンしなければならないわけです。

リターンに関するスタック操作は問(その14)にもありましたが、キチンとスタックの状態を把握さえしていれば難しいことではありません。しかし、プログラムの途中でかりに「RET　Z」などとなっている場合、いったんどこかにジャンプしてからスタックを操作しなければならず、意外と面倒です。

さらに……、「HALF1」がサブルーチンをコールしているとなると、話はますます複雑になります。というのは、そのサブルーチン先でスタック操作が行われている可能性だってあるからです。

……どうやら、このままでは化け猫小説になりそうな雲行きです。もっと簡単な手法で解決することにしましょう。それは、例の書き換えを使う方法です。

```
ADDA1 :   LD      A,3CH          ←3CH=INC A
          JR      ADSB1
SUBA1 :   LD      A,3DH          ←3DH=DEC A
ADSB1 :   LD      (POINT),A
              :
POINT :   INC     A
              :
          RET
```

こうすれば、プログラムを完全共有化できる上、スタックなどの余計な心配をする必要はなくなります。プログラムが長くなればなるほど、このような書き換えによるメモリ節約は効果を発揮してくれます。

A レジスタの値を破壊したくない場合は、単純に退避させてもいいですし、HL レジスタを使ってもいいでしょう。HL レジスタで「POINT」を指定し、「RES　0,(HL)」とすれば「INC　A」に、「SET　0,(HL)」とすれば「DEC　A」となるからです。

これで、めでたく『猫とコンピュータ』が完成するといいですね……。

問その16 フラグで合図を送る

以前、新潟のドラQという者が質問をしたと思いますが、あれはボクの弟です。弟とボクは3才離れており、ボクも去年は大学受験でした。入学と同時にパソコンを買ってもらったので、パソコンに関しては弟と同じレベルです。

ところで、先日弟がした問（その14）への回答について、もう少し詳しく聞かせてください。あの時の回答には、スタック操作の例としてボールと釘との衝突チェックをあげていましたが、説明としては理解できても現実にはピンと来ないのです。

それは、ボールと釘の衝突を判定するサブルーチン「CCCC」から、どうして「MAINP」へ直接もどるのかという疑問です。衝突チェックなんていうのは、プログラム実行時間からすれば瞬時のできごとです。残りの釘のチェックをしたところで、実際にはなんら問題は生じないはずです。

それとも、直接「MAINP」へもどることにより、なにか特別なメリットでもあるのでしょうか。兄として、知りたいと思います。

下宿先のドラQ II（山形）

答

ウ～ム。そこまで突っ込んだ質問がくるとは、こちらとしても不覚にも予測していませんでした。さすが、兄のドラQ IIさんです。

あの時の例は、あくまでもスタック操作の第一歩として出したのですが、今回の質問により問題は衝突チェックのテクニックへまで発展してしまったようです。確かに、実行時間についてだけ考えれば、残りの釘との衝突をチェックしたところで何の支障もありません。だとすれば、「MAINP」へ直接もどる必要もないということになりそうです。

そこで、まずは破裂があったかどうかを「MAINP」へ伝達する方法について考えてみましょう。最初に思いつくのは、破裂したことを示すワークエリアを確保し、そのワークエリアの値を「MAINP」でチェックする方法です。

```
BATWK:   DB      0

MAINP:           :
                 :
         CALL    AAAA        ←衝突チェック
         LD      A,(BATWK)
```

```
        OR      A
        JR      NZ,GOVER        ←破裂フラグが立っていれば
          :                       ゲームオーバーへ
```

おそらく「MAINP」の書式はこのようなものになるはずです。もちろん「BATWK」に破裂フラグを立てるのはサブルーチン「CCCC」の役目です。そして、このようなプログラムであれば「CCCC」から直接「MAINP」へもどる必要もありません。

つまり、サブルーチン「CCCC」は衝突があるないにかかわらず、すべての釘とのチェックをすればいいからです。では、どのような場合に直接「MAINP」へもどったほうがいいのでしょうか。それは「MAINP」が次のようになっている場合です。

```
MAINP :   :
        CALL    AAAA
        JR      Z,GOVER
          :
```

プログラムとしては、見るからにこちらのほうがスッキリしています。破裂を示すフラグはゼロフラグですから、ワークエリア「BATWK」も不要です。もちろん、サブルーチン「CCCC」ではダミーのスタック操作後にゼロフラグを立てなければなりませんが。

```
CCCC :  LD      A,H
        CP      B
        RET     NZ          ←AAAAへもどる
        LD      A,L
        CP      C
        RET     NZ          ←AAAAへもどる
                :           ←破裂のプログラム
        POP     AF          ←ダミー
        XOR     A           ←ゼロフラグを立てる
        RET                 ←MAINPへもどる
```

つまり、残りの釘とのチェックを省くことで、衝突の判定に利用したゼロフラグをそのまま破裂フラグとして利用しているわけです。もしも、最初のような書式であれば、ゼロフラグを立てる代わりに、「BATWK」に１を入れるというプログラムが必要です。

今回はゼロフラグでしたが、このほかにもルーチンからルーチンへの条件の引渡しには、キャリーフラグなどもよく使用されます。フラグを単にその場の条件分岐のためだけでなく、引数代わりに活用することでプログラムの雰囲気はかなり変化してきます。ドラＱ兄弟にも、あるいは少しばかり差がついてしまったかもしれません。



問 その17 範囲のある比較

ガーガー……。エー、聞こえるあるか。どうぞ……。

こちらは地球から遠く離れたアンドロメチャ星雲、メソポチャ星の外星広報担当、ルートカ・ダッペ将軍ある。ヨロシクあるネ。現在、銀河太陽系惑星・地球に向け知能情報収集のため超能力波を送信中。地球の名誉のため答えるあるよろし……。

これまでの情報によれば、地球にもようやくコンピュータが発生したようであるが、まだまだマシン語は普及してないとのことあるね。わがメソポチャ星は、マシン語が唯一の言語あるが、そのレベルは不明ある。

例えば、Aレジスタの値が 5～20の範囲にあるかどうかを調べる場合、われわれは次のように話すあるネ。フラグの使い方の見本みたいあるよ。Aレジスタがその範囲にない時は、キャリーフラグを立てて結果を返すようにしているある。

```
CHECK :   CP    5
          RET   C
          CP    21
          JR    NC,SETCY
          OR    A
          RET
SETCY :   SCF
          RET
```

地球では、どんなもんあるか。教えるよろし……ガー……。

ルートカ将軍（メソポチャ星）

答

ワッ!!

いきなり変な質問が私の頭に飛び込んできたある……。どうやら、地球人としての能力を調査されているみたいあるネ。これは、真剣に考えなくては……。

しかし、プログラムのレベルからすると、入門者と上級者の中間あたりをウロついているような星のようです。というのは、結果をキャリーフラグで返すという高級なテクニックを使っている割には、プログラムがどうも低レベルなのです。

確かに、キャリーフラグのセット／リセットなど、基本的な常識も知っては

います。ところが、それらのテクニックがプログラムに反映されていないのです。少なくとも、このプログラムは次のようにするべきです。

```
CHECK :     CP     5
            RET    C
            CP     21
            CCF
            RET
```

せっかくニモニックには「CCF」というキャリーフラグを反転する命令があるのですから、ここで使わないという手はありません。なかなか「CCF」を使うチャンスなんてないですからね。ここは絶好の使用例といえるでしょう。

さらに、Aレジスタが破壊されてもいい場合には、マイナスの数値はプラスの大きい数値と同じ（問１の図参照）という、マシン語数値独特の特徴が活用できます。

```
CHECK :     SUB    5
            CP     21-5
            RET
```

最初に５を引いてしまうことにより、５未満の数（0〜4）は −5〜−1 、つまり十六進数では FB_H〜FF_H となります。これは結局21以上（５を引く前）の数値ですから、一度のCP命令で範囲のある判定ができるわけです。

ただし、この判定ではキャリーフラグの立ち方が最初の例と反転しています。このプログラムをコールしているほうでも、条件分岐の条件を反転させる必要があるのは言うまでもありません。

こうしてみると、地球のマシン語レベルも国際的、いや宇宙的に通用しそうなレベルにあるではありませんか。どうであるか、ルートカ・ダッペ将軍……？

問 その18

ペアレジスタの値を2倍に

拙者の名は、武蔵。名前が古いせいか、どうも話し方まで古くさいが、その点はご勘弁願いたい。

現代の武蔵はパソコンをこよなく愛し、マシン語を自在に操ろうと日夜キーボードを叩いておる。先んずれば人を制す。これも、永遠のライバル小次郎に先を越されないための知恵である。

とはいえ、現代の小次郎もどこかでパソコンをいじっているに違いない。いずれマシン語で対決する時が来るであろう。時代は違っても、武蔵が小次郎に負けることは許されぬことなのじゃ。

拙者はいま、対決用プログラムの中でHL／DE／BCの各ペアレジスタを2倍にしなければならんのだが、HLレジスタが使用中のためどうもうまくいかん。どうしたものであろうか……。海が荒れよるのォ。

武蔵旅情（玄海灘）

答

パソコンは人を選ばない。しかし、マシン語を話さない者には本当の心を開かないという。1台のパソコンをめぐって、マシン語とマシン語の壮絶な戦いが始まる……。

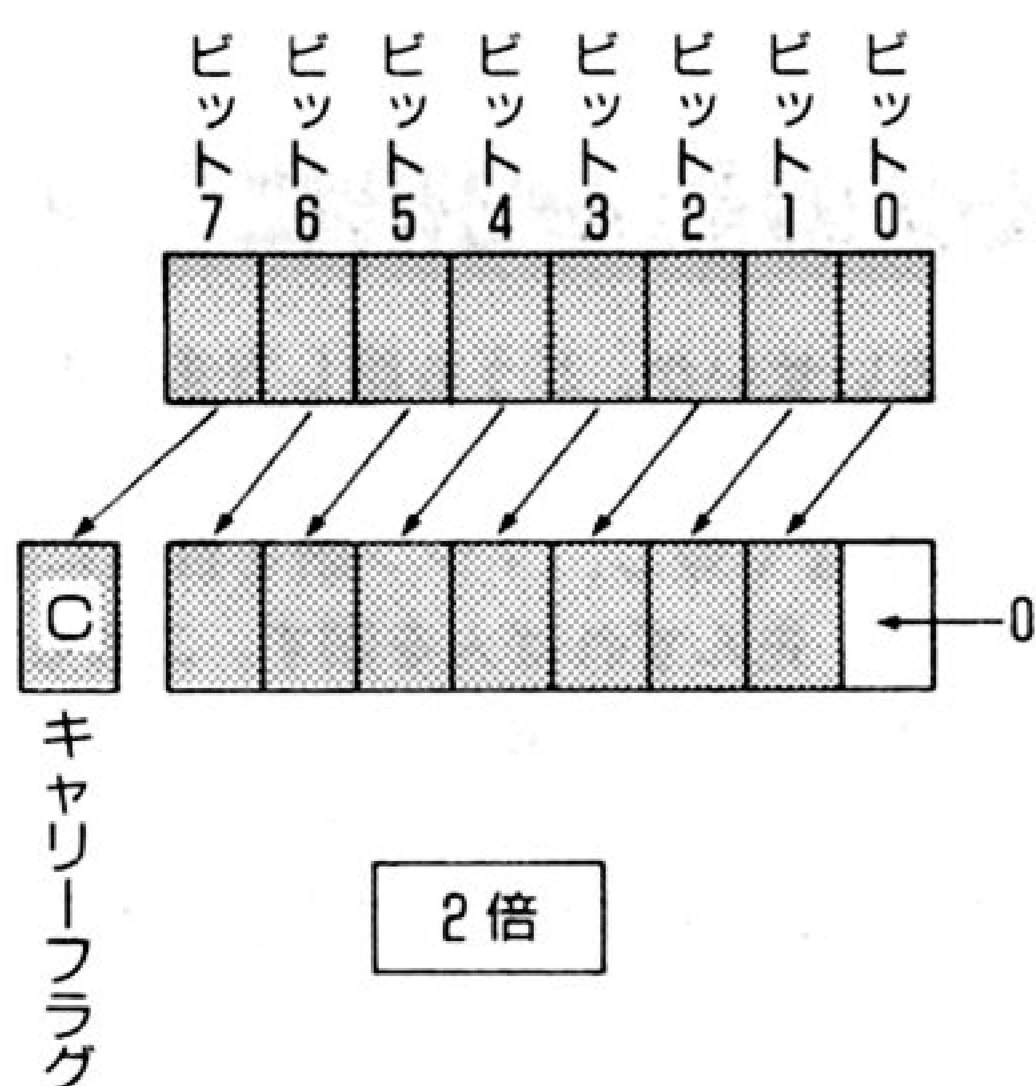
戦いに勝つためには、限られたニモニックの中で、最大限の工夫とヤリクリをしなければならないのです。勝負の世界のキビシサ、それは剣がマシン語に変わっても永遠に変わらぬ心理です。

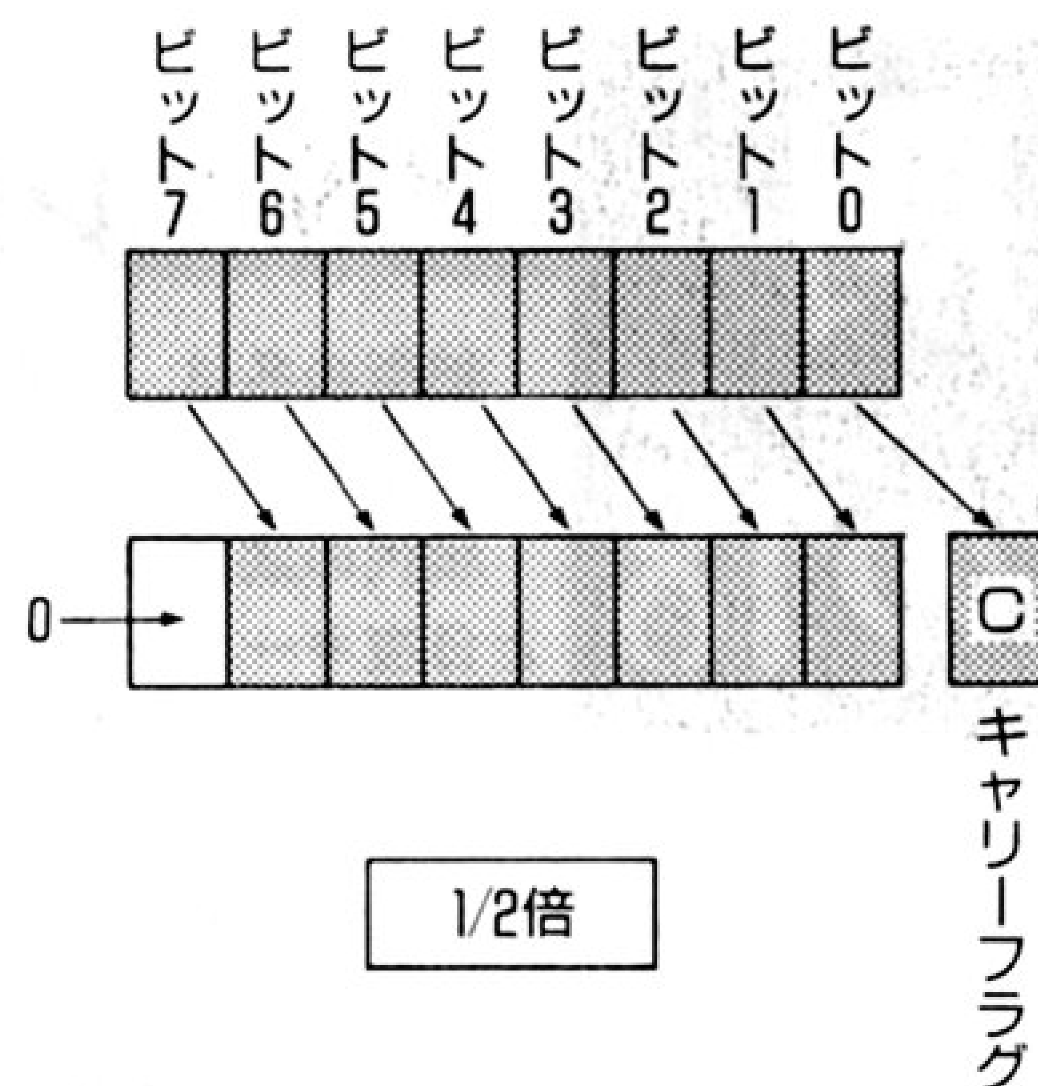
ということで、HLレジスタを2倍するのは簡単です。

```
ADD   HL,HL
```

しかし、レジスタの値を2倍するというのは、なにも同じレジスタどうしを加えるだけではありませんね。そうです。左方向へシフトすればいいのです。その逆に右方向へシフトすれば、レジスタの値は半分になります。

ただし、残念なことに8ビットレジスタのシフト命令はあっても、16ビットレジスタを一度にシフトする命令はありません。そこで、この出っぱったキャリーフラグを利用するのです。





DE レジスタの値を 2 倍にする

```
SLA     E
RL      D
```

BC レジスタの値を 2 倍にする

```
SLA     C
RL      B
```

DE レジスタの値を1/2倍にする

```
SRL     D
RR      E
```

BC レジスタの値を1/2倍にする

```
SRL     B
RR      C
```

たったこれだけのことですが、HL レジスタを介さないで目的を達成したことに意義があるのです。つまり、ペアレジスタに関する命令はほとんどが HL レジスタに関係していますから、HL レジスタが使用中の場合には実に不便なのです。したがって、このような答は簡単であればあるほど利用価値があるというわけです。

でも、これのどこが対決プログラムになるんだろうか……？

問 その19 マイナスの値を1/2にする

遅いぞ、武蔵!!

もはや、拙者の対決プログラムは完成しておる。あとは武蔵の到着を待つばかりなのだが、拙者は到着が遅いからといってイライラしたりはせぬ。拙者だって、過去の話くらいは知っておるからな。

それにしても遅い……。まさか逃げたのでは。とりあえず、最後のプログラムチェックだけでもしておこうか。

ヤヤヤッ!! 暴走してしまった。まだ、プログラムにバグがあるようだ。どうもデータを半分にする部分がマズいようだ。本に載っていたのを、そのまま信用して使ってみたのだが、あの本にはバグでもあるのかも知れぬ……。

拙者の場合、マイナスのデータを半分にしただけなのだが、たしかマイナスとはプログラムを組む者が数値をどう見なすかという問題のはずだ。だから、そこにバグがあるとは思えぬし……。

ちなみに、そのマイナスのデータはAレジスタとDEレジスタについてであるが、次のように本の通りにしている。ウ～ム……わからん。

Aレジスタを1/2にする

```
SRL    A
```

DEレジスタを1/2にする

```
SRL    D
RR     E
```

もしかすると、あの本は武蔵の謀略本なのだろうか……。

ツバメ小次郎（巌流島）

答

すでに武蔵と小次郎の心理戦は始まっているようです。ただ、どちらもプログラムが未完というのが気になりますが……。それにしても、あの巌流島でマシン語とマシン語が火花を散らすなんて、ワクワクしてしまいます。

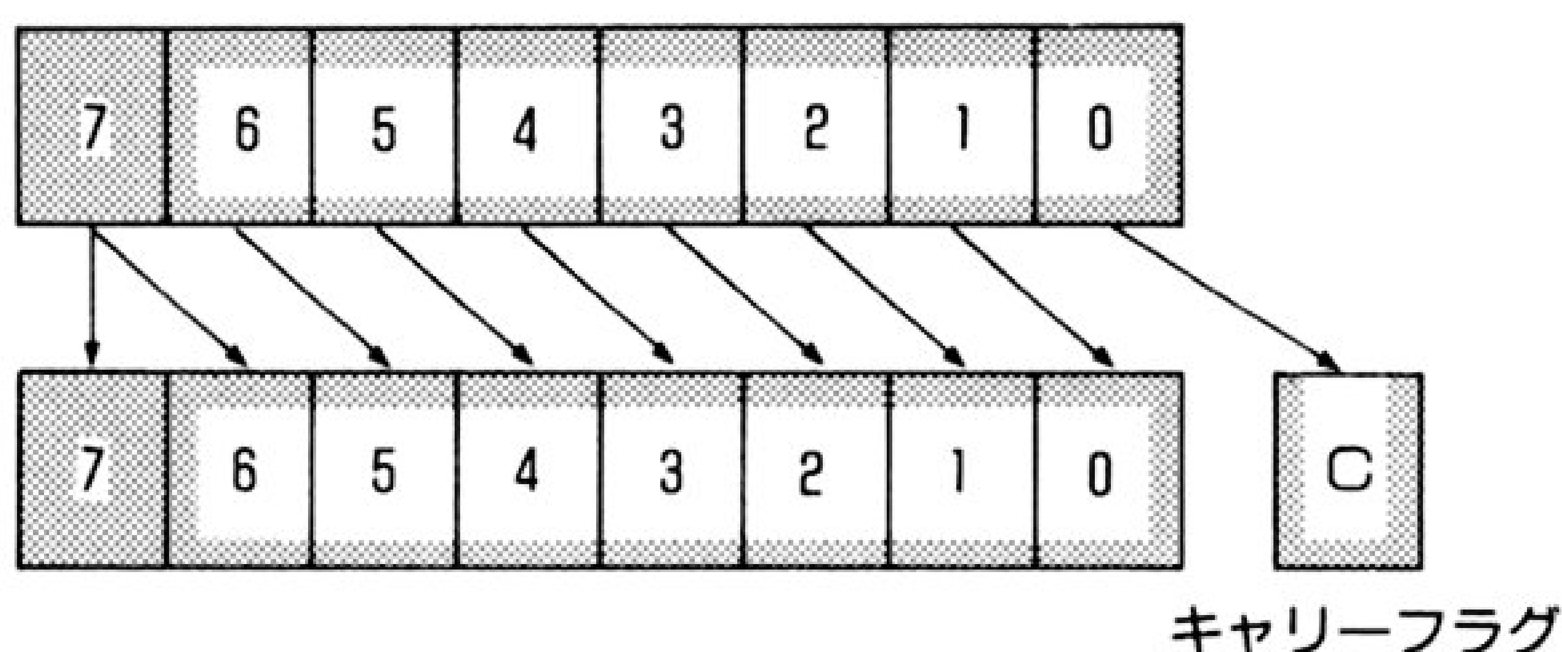
さて、このバグは謀略によるものなどではありません。数値をマイナスと見なすか、プラスと見なすか、それは確かにプログラムを組む人の勝手です。しかし、数値を 1/2にする場合はプラス／マイナスをキチンと区別しなければなりません。問（その18）の方法は、あくまでも数値をプラスと見なしている場合に有効なのです（2倍にするほうはプラス／マイナスを問わない）。というのは、マイナスと見なした数値の割り算（割る数＝2以上の場合）では、必ず最

上位ビット＝1としなければならないからです。わかりにくいので、具体的な例で確認してみましょう。

A＝11111100Bを　252と見なした場合：　A÷2＝　126（01111110B）
A＝11111100Bを　−4と見なした場合：　A÷2＝　−2（11111110B）
A＝00000100Bを　　4と見なした場合：　A÷2＝　　2（00000010B）
A＝00000100Bを−252と見なした場合：　A÷2＝−126（10000010B）

したがって、数値を常にマイナスと考えるのであれば、1/2 したあとで最上位ビットを1にするか、最初に「SCF」としてキャリーが最上位ビットに入るようにローテートすればいいわけです。しかし、数値のプラス／マイナスをサインフラグ（最上位ビット）に従って判断しようという場合は、最上位ビットの値が 1/2後もそのままになるようにしなければなりません。

符号付きの1/2計算



こうすれば、−128〜127（1バイトの場合）、−32768〜32767（2バイトの場合）の数値をプラス／マイナスを問わずに 1/2にシフト演算できます。そして、マシン語にはまさにこのための命令（SRA 命令）が最初から用意されているのです。つまり、プログラムを次のように訂正することで、問題のバグからは解放されるでしょう。

Aレジスタを1/2にする

```
SRA    A
```

DE レジスタを1/2にする

```
SRA    D
RR     E
```

どうやら、武蔵も小次郎も問題が解決した模様です。私はミーハーですから、マシン語とマシン語の凄惨な対決を早くテレビで見たいものです。

問 その20

かけ算を分解して高速化

ＣＱ２メーター、ＣＱ２メーター……。どなたか応答願います。こちらは難破船であります。現在地不明。乗務員一名。

初のパソコンによる自動操舵システム実験のため、栄光の出港をしてからどのぐらいの時がたったのでしょう。残された電源は太陽電池だけとなり、日没とともに自動操舵システムは波まかせとなってしまいます。

おまけに、本船のパソコンによる計算ルーチンにはバグがあったのです。ソースリストを見ると、かけ算ルーチンが空白のままなのです。おそらく、プログラマー氏があとから書くつもりだったのでしょう。コメントとして……、

```
HL = HL x 144
```

と、なっていました。これは、HL レジスタの値を 144倍するというような意味のはずです。でも、それ以上はわかりません。どなたか、このかけ算ルーチンをマシン語で作ってください。このままでは、いずれ幽霊船になってしまいます。

こちらのコールサインは「ＪＪ１ＶＲＱ」です……。

難破船長パフェ（太平洋）

答

「ＪＪ１ＶＲＱ」……？

ち、ちょっと、そりゃ私のコールサインですゾ。まだ一度も使ったことがないのに、どうしてそんなところに……。難破船にナンパされてしまった!?

これは、真剣になってコールサインを取りもどさなければいけません。ニモニックの一覧表（インストラクション表）には、かけ算の命令などありません。すなわち、マシン語でのかけ算は、足し算を繰り返すしかないということです。

```
          EX    DE,HL
          LD    HL,0
          LD    B,144
LOOP 1 :  ADD   HL,DE
          DJNZ  LOOP1
```

これが、もっとも単純に考えられる足し算によるかけ算です。これでも正しい結果は得られますが、ループを使用したかけ算では、プログラムの長さに比べて実行ステート数がかなり長くなってしまいます。

早い話、これはカッコ悪いということです。そこで、144 という数字を次のように分解します。

144＝2×2×2×2＋2×2×2×2×2×2×2

レジスタの値を2倍にするのは簡単です。それを繰り返せば、2倍が4倍、4倍が8倍と、ガマの油売りの前口上みたいに倍々にふくらんでいきます。これをプログラムで実行するのです。

```
ADD   HL,HL      ←2倍
ADD   HL,HL      ←4倍
ADD   HL,HL      ←8倍
ADD   HL,HL      ←16倍
LD    D,H
LD    E,L        ←DE＝16倍したHL
ADD   HL,HL      ←32倍
ADD   HL,HL      ←64倍
ADD   HL,HL      ←128倍
ADD   HL,DE      ←144倍（128倍＋16倍）
```

かける数の決まったかけ算は、このように2の倍数で分解してから計算すると、実行時間が大幅に短縮します。この効果は、かける数が大きくなればなるほど、そしてかけ算をする回数が増えれば増えるほど大きく現れてきます。

では、早くプログラムを直して、「ＪＪ１ＶＲＱ」を返してください。

問その21 ブロック転送

わたしはコンピュータ占い師マーサ・カーイといいます。実は、コンピュータ占いといいましても、これまでのものはほとんどがイカサマでした。あえてネタをばらしてしまいますと、最初にインプットした占いデータを、ただ順番に出していただけなのです。

わたしは、職業がら女装をしていますが、本当は男性です。これも一種のイカサマなのですが、このほうがなんとなく神秘的に見えるのです。

ところが、先日イカサマを見破られてしまったのです。アラ、女装のほうではなくプログラムのほうですワ。ホホホ……。そこで、これからは本格的にコンピュータ占いをプログラム化しようと思い、覚えかけのマシン語でプログラムを組み始めました。

でも、データをブロック転送する際に、時々データがメチャクチャになってしまうことがあるのです。そのブロック転送は、5000_H～$5FFF_H$ 番地の内容を 5100_H番地へ転送するという簡単なものです。

```
LD      HL,5000H
LD      DE,5100H
LD      BC,1000H
LDIR
```

HL レジスタに 5000_H番地をセット、DE レジスタに転送先をセット、そして BC レジスタには転送バイト数をセットしています。どこも悪いところはないはずです。これぱかりは占うこともできず、困ってしまいますわ、ホント。

ニューハーフ占い師マーサ（東京）

答

こういう丁寧な質問をいただきますと、こちらまで女装した気分になってしまいそうですわ、オホホホ……。

なんだか、おかしな空気が漂いだしたようです。このままでは、まともな回答ができなくなりますので、失礼ですが相手を見ないようにして答えることにします。

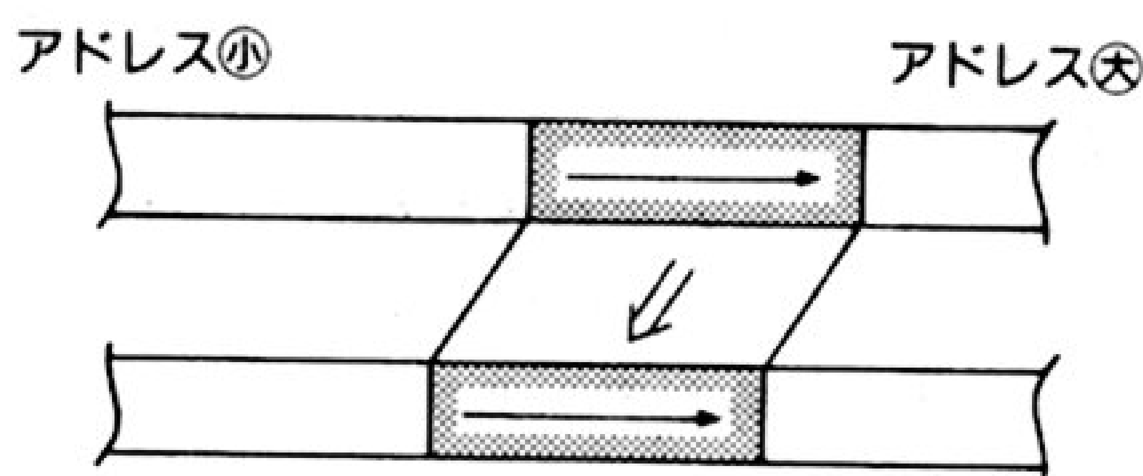
このブロック転送は、一見するとなんのバグもないように見えます。特に、バグがないと思ってしまうと、長いこと悩むことになるほどです。

しかし、ブロック転送において、転送するデータのあるアドレスとその転送先が重なっている場合は、LDIR 命令と LDDR 命令をキチンと使い分けなけれ

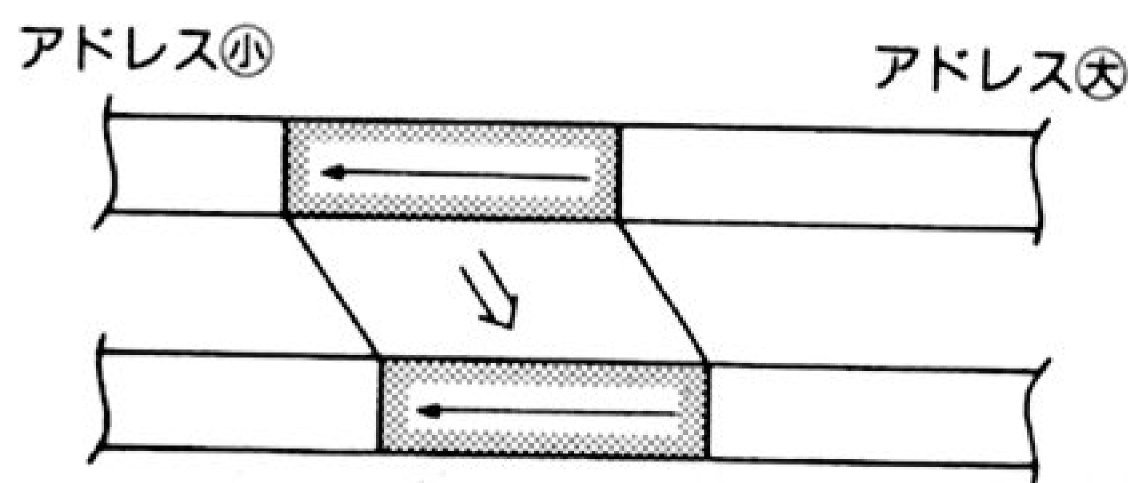
ばなりません。今回の場合も、このままでは 5000_H 番地の内容が 5100_H 番地に転送された時点で、まだ転送し終えていない 5100_H 番地の内容が消えてしまっているのです。

では、どのように2つのブロック転送命令を使い分けるのか、図によって確認してみましょう。

LDIR命令（アドレスの小さい方から順次転送していく）



LDDR命令（アドレスの大きい方から順次転送していく）



転送先がオリジナルのデータと重ならない場合、一般にはLDIR命令が使用されます。それは、いちいち転送データのエンドアドレスを計算しなくて済むからです。そのせいでしょうか、ついついLDDR命令というのは忘れられがちな存在です。しかし、天災とバグは忘れたころにやってくる。それを忘れないでください。

```
LD      HL,5FFFH
LD      DE,60FFH
LD      BC,1000H
LDDR
```

これで、ブロック転送は完璧に行われるはずですわ、オホホホ……。しまった、また相手の顔を見てしまったわ。ワワワ……。

問その22 CP命令でのジャンプ

わたしはナース。つまり看護婦さんです。でも、本当をいうとまだ看護婦見習いなんです。看護婦さんっていうと、白衣の天使なんて憧れる人もいますけど、実際はすごく大変な仕事です。

それはさておき、わたしは看護学勉強のためパソコンを活用しようと思っています。でも、そのためのソフトなど市販されていませんから、プログラムも全部自分で組まなければならないのです。プログラムはマシン語を使っていますが、不慣れのためバグがないということしか自信はありません。

ここに、Aレジスタの値（1～5）によって5個所に分かれるようなプログラムがあります。バグはないと思いますが、誰でもこう組むものでしょうか。

```
          CP    1
          JR    Z,PROG1
          CP    2
          JR    Z,PROG2
          CP    3
          JR    Z,PROG3
          CP    4
          JR    Z,PROG4
PROG5 :          :
```

せめてプログラムだけでも見習いの肩書きが取れるとうれしいのですが。未熟なわたしのプログラムを、どうぞよろしくお願いします。

ホワイトエンジェル（佐賀）

答

ナース……。なんと美しい言葉のひびき。これでは、白衣の天使に憧れて、無理にケガをする人が続出しそうです。ついでに、マシン語のバグを治療してくれるような看護婦さんがいると、マシン語がもっともっと広がるのですが……。

それにしても、バグが出ないだけでは満足せず、さらに上を目指すというファイトはこちらが見習いたいほどです。

実は、このようなプログラムは初心者がよく作ります。しかも、プログラムとしてはバグはありませんから、このままズルズルと中級者になってしまうこ

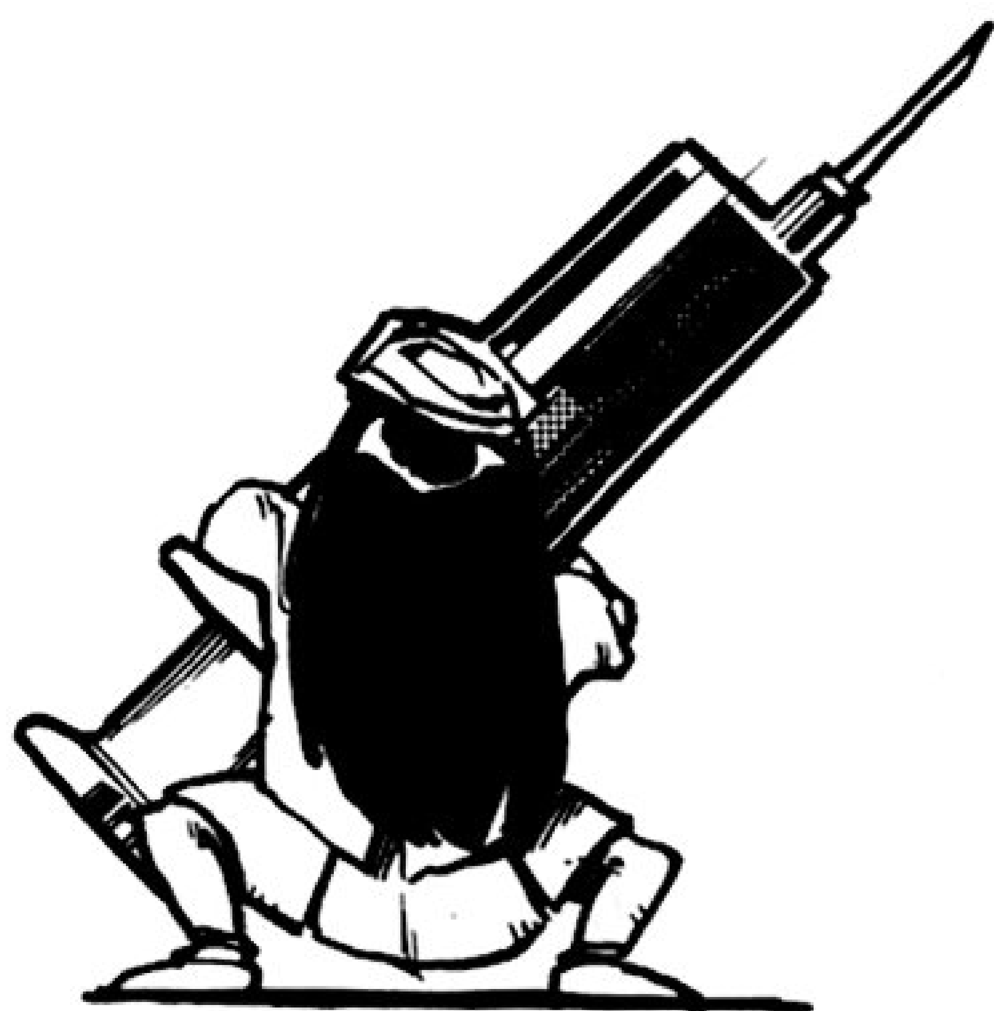
とも少なくないようです。しかし、条件分岐というのはプログラムの重要な拠点ですから、できるだけシンプルに、そして素早く分岐させることが大切です。

この例は、そういう意味では初歩の条件分岐といえるでしょう。ここでは分岐の条件にゼロフラグだけしか使用していませんが、せっかくCP命令を実行したのですから、せめてキャリーフラグまでは活用したいものです。

```
          CP   2
          JR   C,PROG1
          JR   Z,PROG2
          CP   4
          JR   C,PROG3
          JR   Z,PROG4
PROG5 :         :
```

CP命令を使う回数が4回から2回になりました。とりあえず、この程度の条件分岐ではこんなものでしょうが、分岐テクニックにはまだまだ種類があります。プログラムのレベルが上がるにつれ、分岐のレベルも合わせて上げていかないと、プログラムが条件分岐の山になってしまいます。

そんな時にこそマシン語専用のやさしい看護婦さんがいてほしい、と多くの孤独なプログラマーは願っているのです。



問 その23

共通項のあるジャンプ

私は、俗に団塊の世代といわれる中年のオッサンです。特にこれといった趣味はありませんが、まだまだ若いモンには負けるかという気持ちで、パソコンなども頑張ってやっております。

趣味はないと書きましたが、実は趣味に近いような形で学生のころから続けていることがあります。それは献血です。献血回数は、今のところ48回ですが、目標は 100回を超えることです。

献血をすると、しばらくして血液の分析結果が送られてきますが、私はこのデータをパソコンにインプットして健康管理に役立てています。いま、Aレジスタの値によって条件分岐するプログラムがあるのですが、ジャンプ先ではすべてAレジスタの値を8にしなければなりません。

現在のプログラムを簡単に書いてみますが、なぜか無駄があるような気がします。

```
          CP    2
          JR    C,PROG1
          JR    Z,PROG2
          CP    4
          JR    C,PROG3
          JR    Z,PROG4
PROG5 :   LD    A,8
                :
```

以下、PROG1～PROG4 すべて「LD　A,8」で始まります。無駄があるかどうか、そして改良できるかどうか見てください。

中年ベビー（兵庫）

答

同じようなタイプの人間は、アチコチにいるものです。私も、今では献血は趣味みたいなもので、血を抜いてもらうと気分がスッキリするほどです。それに、あのチクリと射された瞬間の痛みは、慣れると快感ですからね……？？？

ところで、献血された血液は一滴も無駄にしないそうですが、このプログラムには想像どおり無駄があります。それが、すべてのジャンプ先にある「LD　A,8」であるということは、誰でも簡単に想像がつくでしょう。しかし、そんな簡単なことでも、いざ解決しようとすると、すぐには名案が浮かばないかも

しれません。あるいは、そんなことは気にしないという人もいるでしょう。
　でも、この本を読んだからには、プログラムの無駄は省くようにしてもらわなければなりません。

```
          LD    B,A
          LD    A,8
          DEC   B
          JR    Z,PROG1
          DEC   B
          JR    Z,PROG2
          DEC   B
          JR    Z,PROG3
          DEC   B
          JR    Z,PROG4
PROG5 :           :
```

　これで、ジャンプ先では「LD　A,8」をする必要がなくなったわけです。では、判断するAレジスタの値に 0が含まれている時はどうでしょうか。
　最初のやり方であれば、「OR　A」としてゼロフラグでジャンプさせるか、「CP　1」としてキャリーフラグでジャンプさせることができます。今回も同様にキャリーフラグが利用できればいいのですが、「DEC　B」ではキャリーフラグは不変です。
　これに対する１つの解決策は、最初の「DEC　B」の前に「INC　B」としてしまうことです。こうすれば、すべて同じようにゼロフラグでジャンプできるからです。また、もう１つの方法はサインフラグを見てジャンプさせる方法です。

```
          :
    DEC   B
    JP    M,PROG0
    JR    Z, PROG1
          :
```

相対ジャンプ（JR）命令には、サインフラグによる条件分岐がありませんか

ら、この場合だけは絶対ジャンプ（JP）命令を使います。また、サインフラグは「DEC　B」の結果、Bレジスタの値が 80_H～FF_H であればすべて立ちますから、この手法でゼロが判定できるのは、Bレジスタの初期値が128以下（0～80_H）の場合に限られます。

したがって、それ以上の条件分岐をしたい場合は、「INC　B」をしてから「DEC　B」でゼロ判定をするしかありません。もっとも、それほどたくさんの分岐がある場合には、分岐方法自体を考慮するのが普通です。

それについては、問（その24）で……。



問その24 テーブルを利用したジャンプ

小話その１：ボクの友人は、50メートル競泳を見て「あれは身長競争だ」と言い張っている。ワケを聞くと「だって、身長50メートルの人間なら飛び込むだけでゴールだ」……。

くだらない……なんて言わないでください。こんな小話を作るのが、わが校自慢の小話クラブです。すでに、名作と自称するものがかなりあります。でも、大変なのはそれを管理することです。とはいえ、そこは高校のクラブです。なんと、名作小話をパソコンで管理することになったのです。

……ということで、プログラマーはパソコン所有者というだけの理由でボクの役目となりました。もちろん、カッコつけてマシン語でプログラムを組んでいます。

いま、0〜99 までの小話に簡単なアニメをつけようとしています。そのため、小話の番号別にそれぞれのルーチンへジャンプさせなければなりません。

```
CP    1
JP    C,ANI00
JP    Z,ANI01
CP    3
JP    C,ANI02
JP    Z,ANI03
   :
```

こんな感じで各アニメ処理ルーチンへジャンプさせようとしていますが、どんなもんでしょうか。小話同様、名作と言われるようなプログラムにしたいと思います。

クラブ小話（和歌山）

答

同じような小話をひとつ……。

食事中に、母が肥満ぎみの父に向かって「理想の体重は身長から 110を引いた値よ」と言った。それを聞いていた小学１年生の息子が、突然食べるのをやめた。ちなみに、息子の身長は 110cm だった……。

クラブ小話……なんだか、銀座のクラブと間違えそうな名前ですが、一応は高校のクラブと信じておきましょう。でも、この質問には銀座のクラブ「小話」のほうが解決しやすいのです。というのは、テーブルを使うからです？？？

もちろん、ここでいうテーブルとはジャンプ用テーブルのことです。とりあえず、サンプルとしてテーブルには10のジャンプ先を用意してみました。

```
TABLE :   DW     ANI00,ANI01,ANI02,ANI03,ANI04
          DW     ANI05,ANI06,ANI07,ANI08,ANI09

ATJMP :   LD     HL,TABLE
          RLCA
          LD     E,A
          LD     D,0
          ADD    HL,DE
          LD     E,(HL)
          INC    HL
          LD     D,(HL)
          EX     DE,HL
          JP     (HL)
```

これが、テーブルジャンプの一般的な用法です。ジャンプ先を増やすには、このテーブルにジャンプ先を追加するだけです。プログラムも短くて済む上、どのルーチンへジャンプするのも実行時間が変わらないというのが特徴です。

このプログラムでは、Aレジスタの値が 0〜127 までという条件がつきますが、もしAレジスタをフルに使用したい場合は次のようにすればＯＫです。

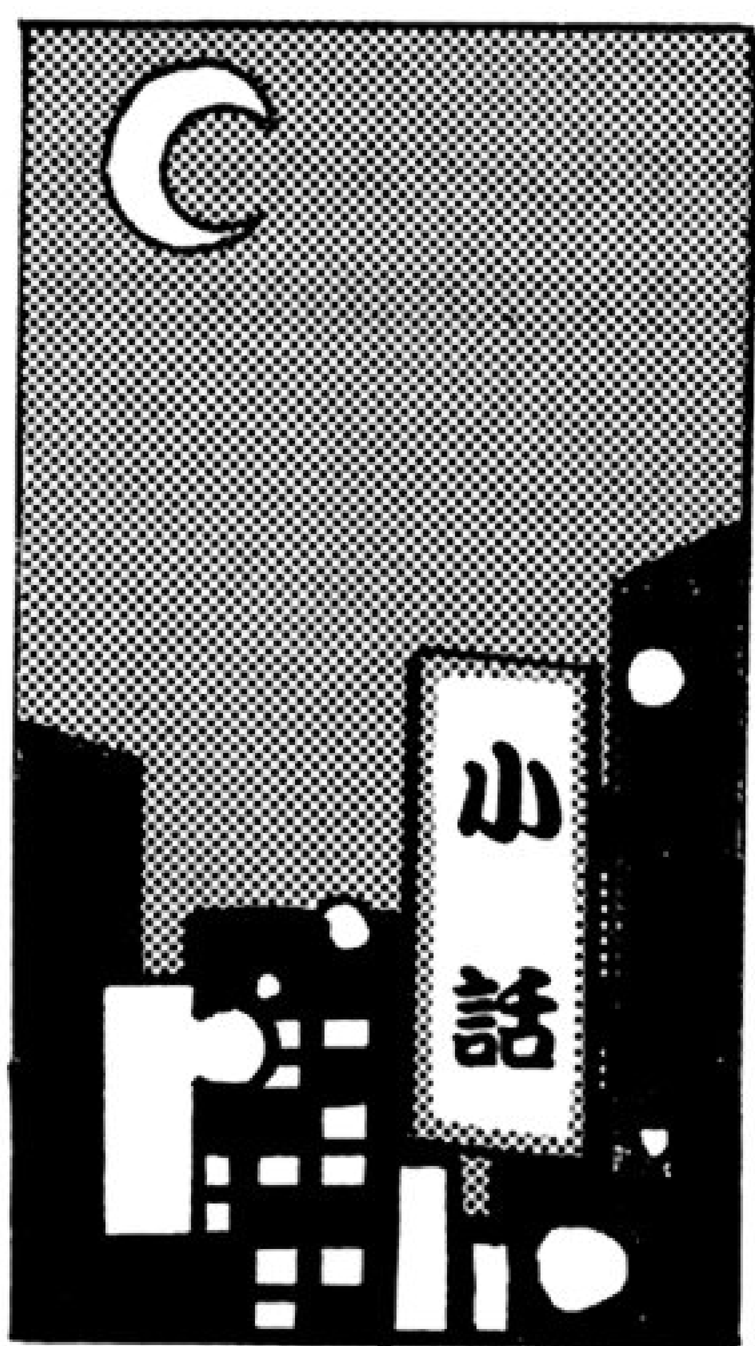
```
ATJMP :   LD     L,A
          LD     H,0
          ADD    HL,HL
          LD     DE,TABLE
          ADD    HL,DE
          LD     E,(HL)
          INC    HL
          LD     D,(HL)
          EX     DE,HL
          JP     (HL)
```

ほとんど同じといえないこともありませんが、常に最少バイト・最短ステー

トを考えるのがマシン語のおもしろさの一面ですから、あえて分けてあるのです。

さて、プログラムの最後に「JP　(HL)」という命令があります。普通は（HL）というとHLレジスタの指し示すアドレスの中身という意味ですが、この命令の場合はHLレジスタが示すアドレスへのジャンプとなります。

同様の表現で、「JP　(IX)」「JP　(IY)」という命令がありますが、これもインデックスレジスタの指し示すアドレスへのジャンプです。こういう紛らわしいカッコはないほうがいいのですが、きっとカッコつけたかったんでしょう……？？？



問その25 スタックを利用したジャンプ

ホップステップジャンプで、こちらは南海の孤島で孤独な生活を送っているナゾの漂流者です。電気もない、ガスもない、水道もない、ナイナイずくしの中で、唯一の楽しみは頭を使った「想像マシン語」です。

時間だけはタップリとあるので、そのうち「想像マシン語」による「想像ゲーム」ができるはずです。でも、「想像デバッグ」を想像するだけでメゲそうです。

最近、覚えたばかりのテーブルジャンプを重宝して使っていますが、HL も IX も IY も使用中のルーチンがあります。だから、「JP　(HL)」も「JP　(IX)」も「JP　(IX)」もできません。

どうやって、目的地のテーブルへジャンプしたらいいでしょうか。

半魚人（沖の鳥島）

答

ち、ちょっと沖の鳥島といったら、このあいだ話題になった水没寸前の岩じゃないですか。どうやって、そんな島へ……!?

盲将棋というのは聞いたことがあるけど、「想像マシン語」による「想像ゲーム」というのは初耳です。それが完成した時には、本物の人間コンピュータの誕生として大ニュースになるでしょう。

テーブルを用いてジャンプするのに、「JP (HL)」も「JP (IX)」も「JP (IY)」も使えないということですが、これは詰将棋でいうならちょっとヒネった一手詰めといったところです。問(その24)を例にしてプログラムを組んでみましょう。

```
ATJMP :  PUSH   HL
         LD     HL,TABLE
         RLCA
         LD     E,A
         LD     D,0
         ADD    HL,DE
         LD     E,(HL)
         INC    HL
         LD     D,(HL)
         POP    HL
```

```
        PUSH    DE          ←「JP　(DE)」を実現するため
        RET
```

これはスタックの操作の一種なのですが、DE レジスタの値をスタックに入れ、RET 命令によりそのアドレスにもどるようにジャンプするものです。このテクニックを使うと、結構色々な形のジャンプができそうですね。

では、A レジスタも含めて空いているレジスタがまったくない場合はどうしたらいいでしょうか。こんな時にも、さりげなくエレガントにスタックの中身をリターンアドレスとすることができます。

```
ATJMP :  PUSH    HL
         PUSH    AF
         LD      HL,TABLE
         RLCA
         ADD     A,L
         JR      NC,ATJP1
         INC     H
ATJP1 :  LD      L,A
         LD      A,(HL)
         INC     HL
         LD      H,(HL)
         LD      L,A
         POP     AF
         EX      (SP),HL
         RET
```

テーブルアドレスの計算には、問（その7）にあるテクニックを使っています。リターン直前の「EX (SP), HL」で、HL レジスタにあるジャンプ先がスタックに入り、最初に退避した HL レジスタの値と入れ換わります。「PUSH HL」を、さりげなく活用している点に注目してください。

そろそろ、救援船にスタックジャンプしたらどうですか。天才半魚人さん!!

問その26 ビット別のジャンプ

映画ってスバラシイですね。映画を見るたびにそう思わずにはいられません。涙と笑いを誘いながら、夢と感動を与えてくれるのです。そして、映画館から外へ出た時の太陽のまぶしさと現実の鼓動、このギャップはテレビでは絶対に味わえないものです。

ところで、映画にはよく実在する町がでてきます。そして、映画の町こそが本物の町のような錯覚を覚えます。でも、日本にはカサだけを売っている傘屋さんなんてあるでしょうか。私は見たことがありません。

それなのに、『シェルブールの雨傘』の舞台は傘屋さんです。もし実在するなら、あんな町のあんなお店で傘を買ってみたい……。

そんなことを考えながら、プログラムを組むのが私の楽しみです。いま、Aレジスタにはビット別に意味を持たせた値が入っています。つまり、8種類のフラグになっているわけですが、これをビット別に各ルーチンへジャンプさせたいのです。

```
MAINP :  RRCA
         JR     C,PROG0     ←ビット 0=1 なら PROG0 へ
         RRCA
         JR     C,PROG1     ←ビット 1=1 なら PROG1 へ
         RRCA
         JR     C,PROG2     ←ビット 2=1 なら PROG2 へ
         RRCA
         JR     C,PROG3     ←ビット 3=1 なら PROG3 へ
         RRCA
         JR     C,PROG4     ←ビット 4=1 なら PROG4 へ
         RRCA
         JR     C,PROG5     ←ビット 5=1 なら PROG5 へ
         RRCA
         JR     C,PROG6     ←ビット 6=1 なら PROG6 へ
         RRCA
         JR     C,PROG7     ←ビット 7=1 なら PROG7 へ
         RET
```

ジャンプの優先権はビット 0から順番に低くなっていきます。そして、このようにビット別にジャンプをさせるケースが数カ所あります。こういう場合はテーブルを利用したジャンプは無理でしょうか。

喫茶シェルブール（長崎）

答

シェルブールとはどんな町なのだろう。町には音楽があふれ、人々は色とりどりの傘をさしながら歩いている……。

『シェルブールの雨傘』を見た人ならそう思って当然ですよね。私も、その美しさに憧れて、わざわざシェルブールまで行ってきました。できたら、おみやげに傘でも買おうと思いながら……。

小さな港町には、にぎわいも音楽も傘屋もありませんでした。観光客もいない、ただの田舎町です。ひょっとすると会えるかも、と思ったカトリーヌ・ドヌーブ……いるわけがありません。

でも、なぜかホッとしました。みやげもの屋なんてあったらガックリきたでしょう。ただの町だからこそ、シェルブールは永遠にすばらしいのです。少し古い情報ですが、今でもそんな素朴な町だと信じています。

さて、このようなプログラムでもテーブルによるジャンプは可能です。

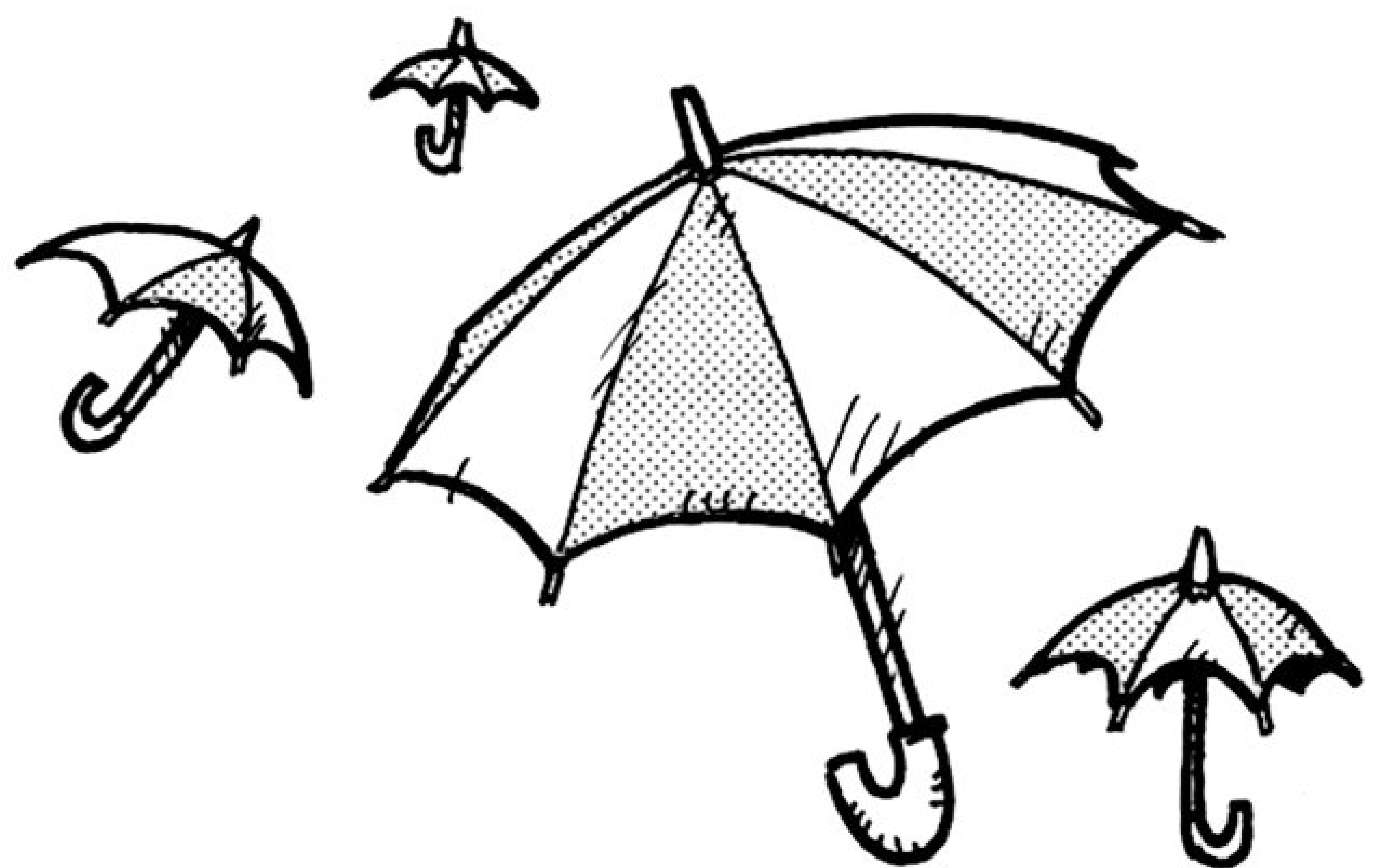
```
TABLE :  DW      PROG0,PROG1,PROG2,PROG3
         DW      PROG4,PROG5,PROG6,PROG7

BTJMP :  LD      HL,TABLE
JUMP8 :  LD      B,8
JMP8L :  RRCA
         JR      C,HLJMP
         INC     HL
         INC     HL
         DJNZ    JMP8L
         RET
HLJMP :  LD      E,(HL)
         INC     HL
         LD      D,(HL)
         EX      DE,HL
         JP      (HL)
```

一見すると、これは質問にあったプログラムより複雑で、時間もメモリも余計に消費しそうな感じがするでしょう。それは事実であり、あえてこのようにする必要がない場合も多いかもしれません。

しかし、このプログラムの特徴は、複数のテーブルがあってもプログラムを共通して使用できる（JUMP8 以下）という点にあります。また、ジャンプ先をテーブル上で管理できるので、変更やデバッグが非常に楽になります。

七色の雨傘ならぬ、八色のテーブルジャンプとして活用しましょう。



問その27 2つのプログラムを共有する

ヒッヒッヒッヒ……。

わたしゃ魔女じゃよ。ホラ、あの黒いとんがり帽子をかぶって、魔法のホウキにまたがって空を飛ぶ魔女じゃ。有名じゃから、知っておるじゃろ。

でも、この魔法のホウキが、実はマイクロコンピュータに制御されていたなんてことは知らんじゃろ。このホウキの秘密は、この長い柄にあるのじゃ。この柄にはいくつかのバージョンがあってな、魔女のランクによって交換されるシステムなのじゃよ。

わたしゃ、ランクの低い三等魔女じゃが、秘かにマシン語を覚えて、この柄のプログラムを変えてしまおうと思っているのじゃ。

そこで質問じゃ。柄のほうのプログラムは何とかするとして、問題は動力部にあたる刷毛（はけ）とのヤリトリをどうするかじゃ。勝手に柄のプログラムをいじると、刷毛が柄のプログラムをコールしている場合に困るじゃろ。

つまり、バージョンの違う複数のプログラムを相互に利用し合うにはどうしたらいいじゃろか、というのが質問なのじゃ。

魔女だからといって、イジワルはしないでおくれ……。

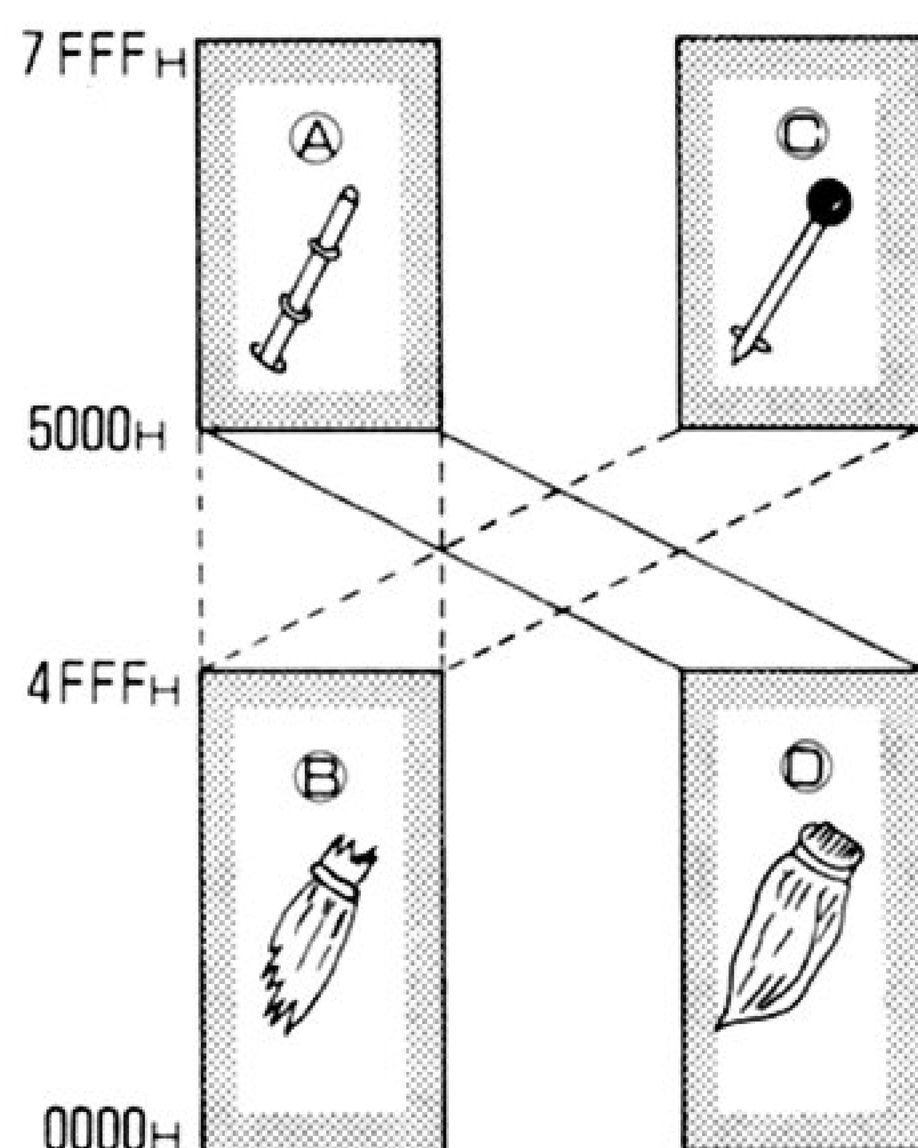
西洋の魔女（ノートルダム）

答

確か、ノートルダムに住んでいるのは‘せむし男’のはずでは……？　それにしても現代の魔女はマシン語まで勉強しなければならないとは、本当にご苦労なこととしか言いようがありません。

質問があまりにも魔女的なので、もう少し普通の人間にもわかるように、問題を具体化してみましょう。例えば、Ⓐというプログラムが 5000_H～$7FFF_H$ にあるとします。ちなみに、魔女さんの話でいえば、これが柄のプログラムです。そして、Ⓑというプログラムが 0000_H～$4FFF_H$ に存在していると考えてください。これが、すなわち刷毛用のプログラムです。

ⒶとⒷが単なる連続したプログラムであれば、ⒶからⒷのプログラムをコールしたりジャンプするのは自由です。同様にⒷからⒶのプログラムを利用することも自由です。しかし、状況によってⒶのプログラムがⒸというプログラムに変わるとしたら、あるいはⒷのプログラムがⒹにバージョンアップしたらどうなるでしょうか。



プログラムが変更になったり、バージョンが変わったりすれば、サブルーチンのアドレスも当然変化していると考えなければなりません。そうなっても問題がないようにするには、たとえプログラムが変化しても、サブルーチンのエントリーアドレスが変化しないようにすればいいわけです。

これに対する１つの解決策は、最初からサブルーチンのメモリ領域にゆとりを持たせておくことです。こうすれば、少しくらいプログラムを変更しても、そのゆとり内で処理することができるからです。

しかし、この方法には無駄なメモリ消費が多い上、サブルーチンが多くなると開発そのものに支障をきたします。常にアドレスを考えながらプログラムを組まなければならないし、ゆとりの範囲内に新しいプログラムが納まらなければ、メモリの空きを探してアチコチにジャンプさせなければなりません。第一、プログラムを組むのに、プログラム以外のことに神経を使うのは不経済です。

そこで、互いのプログラムの先頭部分にジャンプテーブルを作り、相手のサブルーチンを利用する際には、そのジャンプテーブルに対して行うようにするのです。

プログラムⒶ Ⓒ

```
        ORG   5000H

BBBB0 : EQU   0000H
BBBB1 : EQU   0003H
BBBB2 : EQU   0006H
BBBB3 : EQU   0009H
BBBB4 : EQU   000CH

        JP    AAAA0
        JP    AAAA1
        JP    AAAA2
        JP    ADUMY
        JP    ADUMY
```

プログラムⒷ Ⓓ

```
        ORG   0

AAAA0 : EQU   5000H
AAAA1 : EQU   5003H
AAAA2 : EQU   5006H
AAAA3 : EQU   5009H
AAAA4 : EQU   500CH

        JP    BBBB0
        JP    BBBB1
        JP    BBBB2
        JP    BDUMY
        JP    BDUMY
```

```
ADUMY :    RET
              ⋮
```

```
BDUMY :    RET
              ⋮
```

この例では、それぞれ3つずつ共通ルーチンを使用し合っており、予備として2つのダミージャンプ先を確保しています。こうしておけば、将来バージョンアップした場合、5つまで共通ルーチンを利用することが可能です。ジャンプテーブルは必要に応じていくらでも増やすことができます。

利用の仕方は簡単ですね。例えば、Ⓐから Ⓑの「BBBB2」をコールしたい場合、そのまま「CALL　BBBB2」とすればいいのです。アセンブル作業は、それぞれ独自に行いますから、たとえⒶのプログラムを修正して共通ルーチンのアドレスが変わっても、Ⓑのプログラムには何ら影響を与えることはありません。

おそらく、魔女ホウキの柄と刷毛の関係もこのようになっているはずです。うまくいけば、三等魔女のホウキも一挙に一等魔女のホウキに化けられるでしょう。



問その28　DJNZ命令の特徴

みゃー……。

おみゃー、ナーゴヤいうたらエーところだでヨー。きしめんはウミャーし、ういろうもウミャーし、大阪と東京のエーところを取り入れた日本の中心都市だでヨ。それに、もうひとつの名物はパチンコ屋だで。

実は、わしゃパチンコが好きで好きで、そーれでナーゴヤに越して来たばかりの老人だでヨ。だから、これはテレビで覚えたインチキのナーゴヤ弁じゃ。テレビじゃ、ミャーミャー言うとったけど、実際にはそうでもないみたいだで。

それはそうと、わしのパチンコ仲間で出る台の番号をパソコンで研究しておるやつがおるでヨ。わーしはそのソースリストを持っとるでな、プログラムの無駄をなくそう思って一部直してやったでヨ。

```
          :
LOOP1 :  LD    A,(HL)
         CP    5
         RET   C
         CP    30
         CCF
         RET   NC
         INC   HL
LOOP2 :  DEC   B
         JR    NZ,LOOP1
         RET
```

⇨

```
          :
LOOP1 :  LD    A,(HL)
         CP    5
         RET   C
         CP    30
         CCF
         RET   NC
         INC   HL
LOOP2 :  DJNZ  LOOP1
         RET
```

以来、出る台がさっぱり当たらなくなったでヨ。わしゃ、なんか悪いことでもしたじゃろか……？

パチンコ老人（愛知）

答

最近はパチンコ屋といっても、半分くらいはスロットマシンが置いてあるようです。やはり、パチンコが一番面白かったのは、左手で1つずつ玉を入れながら打つ古き時代じゃなかったでしょうか。

あの頃は、神技のような早打ちができる名人がいたり、ひとつぶの玉を右手で入れては右手ではじくヒマつぶしの老人もいました。それに比べると、現代はスピードの時代。軍資金も玉もアッという間になくなります。

おっと、こんな話は18禁の方には関係のないことですね。あくまでも、問題はプログラムの内容についてです。

修正したプログラムは、オリジナルより１バイト少なくなり、実行に要する時間も１ループにつき３ステート少なくなっています。もちろん、命令を実行する過程はまったく同じです。しかし、このプログラムは実行過程よりも、実行結果に意味があるプログラムなのです。最後のフラグまで見落としてはなりません。DJNZ 命令はフラグ変化をしないのが特徴です。

まず、このプログラムの目的ですが、HL レジスタで示されるアドレスの中身を、最大でＢ回（アドレスを＋１しながら）チェックし、その結果をフラグで返すことです。オリジナルのフラグ結果と、修正後の結果を比較してみましょう。

(HL)を最大Ｂ回チェック	オリジナル	修正後
(H)　＝ 0～4	CF＝1　ZF＝0	CF＝1　ZF＝0
(HL)　＝ 5～29	CF＝0　ZF＝0	CF＝0　ZF＝0
(HL)　＝ すべて 30以上	CF＝1　ZF＝1	CF＝1　ZF＝不定

Ｂ回のチェックの間に（HL)＝ 0～29となれば、フラグ結果はどちらも同じです。しかし、最後まで (HL) が 30 以上の値であった場合には、修正プログラムではゼロフラグが不定です。不定といっても、セロフラグが１になるのは最後の (HL) の値が 30 の時だけで、それ以外はすべてゼロフラグは立ちません。

したがって、せっかく最後まで 30 以上であっても、RET 時には途中に 0～4 があったというフラグ結果になってしまうわけです。これでは、このプログラムの価値も不定になってしまうはずです。

今回は、フラグ変化がないという理由で DJNZ 命令が使用できませんでしたが、その逆にフラグ変化がないという特徴を利用することもあります。DJNZ 命令は単にループに便利というだけでなく、このような使い分けをすることも活用したいポイントなのです。

なお、このサブルーチンをコールしたほうのプログラムでは、結果による条件分岐の順序を間違えないように気をつけてください。先ほどのフラグ結果表を見ながら、どの条件で分岐させるべきが確認しておきましょう。

それにしても、なんだか危ないプログラムです。でも、もし実在するなら、ターカク売れそうだでヨ……。



問その29 P/V フラグとは

ハロー‼ここはニューヨークです。父の仕事の都合で、半年前にこちらへ家族そろって引っ越してきました。広大な国アメリカは、なんでもビッグです。アイスクリームもステーキも日本の倍はあります。

家では日本語、外では英語、そして夜はマシン語の毎日です。いま、フラグの利用法について色々研究しています。ところで、よく内容のわからないフラグがあります。それはパリティフラグです。

まだ、実際にこのフラグを利用する機会がありませんが、命令と変化の内容が頭の中で整理できません。当然、具体的な利用方法も思い浮かびません。おまけに、ニモニック表のパリティフラグの項には「IFF」なんていう不可解な文字もあります。

英語で「IF」ならわかります。「もしも～なら」の「もしも」です。もしも、この意味が英語の「もしも」なら、いったい何が「もしも」なのか教えてください。

電話で聞こうとも思いましたが、もしももしもしのもしともしものもしもが混乱するとわからなくなるので、電話はやめました。よろしく‼

現地名・早口トム（ニューヨーク）

答

電話じゃなくても、もう十分に混乱してます。もしもしもも……。読むだけでメゲそうです。もしも電話だったら「もしもし」の「もし」と「もしも」の「もしも」を理解するだけで……アッ、読めたようです。

では、最初にこの「IFF」から解決しておきましょう。これが未解決だと、最後まで頭から「もしも」が離れませんから。

IFF ＝ Interrupt Flip-Flops

これは「もしも」ではなく、割込みのオン／オフの状態を示すものです。つまり、割込み禁止の状態なら「IFF ＝ 0」、割込み許可の状態なら「IFF ＝ 1」となります。ニモニック表で「IFF」となっているのは、「LD A,I」と「LD A,R」の２つの命令です。この命令を実行するとパリティフラグは次のように変化します。

DI（割込み禁止）時：パリティフラグ＝ 0

EI（割込み許可）時：パリティフラグ＝ 1

ただし、このような使い方が必要になることは、ほとんどありません。パリティフラグはニモニック表でP/Vフラグと書かれているように、正式にはパリティ／オーバーフロー・フラグと呼ばれ、命令により次のように異なった反応を示します。

(1) パリティフラグとしての役目（P/Vの項目がPとなっている命令）

おもに論理演算やシフト命令で反応し、データ中のセットされているビットの総数が偶数の場合 1（奇数の場合 0）となります。例えば、AND命令の実行後アキュムレータの値が 00111111B（= $3F_H$）となれば、セットされているビット数は6ケですからパリティフラグ＝1となります。

(2) オーバーフローフラグとしての役目（P/Vの項目がOとなっている命令）

おもに加減算命令で反応し、補数が限界を超えた時に1となります。補数とは、レジスタ値の最上位ビットを＋／－の符号とするものです（問その1参照)。したがって、補数が限界を超えるということは、最上位ビットの値が反転することです。

例えば、8ビットの計算を例にとると 125に4を加えると 129となり、ビット7の値は 0から 1に変化します。つまり、補数表現では －127 となり、＋／－の符号を考えた計算の限界を超えるわけです。このような時に、オーバーフローフラグ＝1となるのです。

このほかにも「LDI」や「CPI」命令などを実行した後、BCレジスタの値が0以外の時にもP/Vフラグ＝1（0の時にはP/Vフラグ＝0）となります。

結局、P/Vフラグというのは命令によって内容の違った反応を示す、いわばフラグの便利屋さんです。しかし、利用する場合の命令は、その意味にかかわらず次のように統一されています。

P/Vフラグ=0 ……「JP　PO,xxxx」「CALL PO,xxxx」「RET　PO」
P/Vフラグ=1 ……「JP　PE,xxxx」「CALL PE,xxxx」「RET　PE」

アメリカにも日本人向けの本屋さんはたくさんあるし、日本の書籍は数日遅れで手に入るそうですから、もう「もしも」で「もしもし」する必要はなくなったでしょう。

問 その30 基礎的な疑似乱数

こちらは、第99次南極観測越冬隊の親衛隊です。越冬隊員に憧れて、秘かに南極までやって来ました。昼間はペンギンと戯れ、夜は全員一丸となってパソコンの前に座っています。その目的は、いつ襲ってくるかわからないブリザードを予測するプログラムを組むことです。

しかし、はっきり言ってこれはゲームみたいなものです。というのは、ブリザードを色々と検討した結果、ほとんど乱数に近い出現という結論に達したからです。それはそれでいいのですが、誰もマシン語で乱数を発生させる方法がわかりません。

ブリザードに襲われるのは、まったく乱数というわけでもなく、大きな周期もあるようです。だから、あまり完全無欠の乱数というのもマズイのです。こんな都合のいい乱数なんてあるでしょうか。

アッ、ブリザードに襲われて停電になってしまった……。

ペンギン野郎（南極）

答

南極からの質問……？

どうやら、この質問はモールス信号で送られてきたようです。そもそも、このモールス信号というのは、わからない人にはピッピッピの乱数にしか聞こえません。それでも困った時のために、SOS（トトト ツーツーツー トトト）だけは覚えておくようにと、子供のころよく言われたものです。

例えば、船倉に閉じ込められた時とか、秘密基地に誘拐された時とか……。でも、そんな経験があるはずもなく、いまだに使ったことはありません。

さて、おたずねの乱数ですが、一般的にコンピュータで乱数といっているのは、本物の乱数ではなく疑似乱数です。これは簡単な計算式によって、一定の周期でランダムに近い数値が得られるようにしたものをいいます。

なんだニセモノか、などとバカにしてはいけません。ゲームなどには、かえってこのほうが適しているのです。というのは、本物の乱数には 0が何回も連続するとか、しばらくの間は10以下ばかりとか、本物がゆえに乱数として好ましくない現象が起きることがあるからです。

その点、疑似乱数はいかにも乱数という感じでバラついてくれます。ここに紹介する乱数は、コールするたびにAレジスタにランダムな数値を入れて返す

という、もっとも簡単で一般的な乱数ルーチンです。

```
RND1 :  LD      A,0
        LD      E,A
        ADD     A,A
        ADD     A,A
        ADD     A,E
        INC     A
        LD      (RND1+1),A
        RET
```

プログラムの内容は、前回の値を5倍し、さらに1を加えたものを新乱数とするというものです。最後に加える数値（ここでは1）は、0以外であれば何でも構いません。この乱数がどのようにループするのかを示しますので、乱れ方の特徴を確認してください。

```
01 06 1F 9C 0D 42 4B 78 59 BE B7 94 E5 7A 63 F0
B1 76 4F 8C BD B2 7B 68 09 2E E7 84 95 EA 93 E0
61 E6 7F 7C 6D 22 AB 58 B9 9E 17 74 45 5A C3 D0
11 56 AF 6C 1D 92 DB 48 69 0E 47 64 F5 CA F3 C0
C1 C6 DF 5C CD 02 0B 38 19 7E 77 54 A5 3A 23 B0
71 36 0F 4C 7D 72 3B 28 C9 EE A7 44 55 AA 53 A0
21 A6 3F 3C 2D E2 6B 18 79 5E D7 34 05 1A 83 90
D1 16 6F 2C DD 52 9B 08 29 CE 07 24 B5 8A B3 80
81 86 9F 1C 8D C2 CB F8 D9 3E 37 14 65 FA E3 70
31 F6 CF 0C 3D 32 FB E8 89 AE 67 04 15 6A 13 60
E1 66 FF FC ED A2 2B D8 39 1E 97 F4 C5 DA 43 50
91 D6 2F EC 9D 12 5B C8 E9 8E C7 E4 75 4A 73 40
41 46 5F DC 4D 82 8B B8 99 FE F7 D4 25 BA A3 30
F1 B6 8F CC FD F2 BB A8 49 6E 27 C4 D5 2A D3 20
A1 26 BF BC AD 62 EB 98 F9 DE 57 B4 85 9A 03 10
51 96 EF AC 5D D2 1B 88 A9 4E 87 A4 35 0A 33 00
```

また、最後に1を加えないとどうなるか、あるいは5倍ではなく2倍とか3倍にして1を加えたらどうなるか……。プログラムを少し変更するだけでテストできますから、ぜひ試してみてください。乱数の乱れを改めて認識できるでしょう。

でも、こんなんで本当にブリザードの予測ができるのかどうか、少しばかり疑問が残ってしまいます。

問その31 実用的な疑似乱数

一生のうち、一度でいいから宝くじの一等に当たってみたい。そう思って20年。もう何枚のハズレくじを買ったことでしょう。当たるのはいつも末等ばかり……。

末等だけは、10枚買えばイヤでも当たります。だから、あれはハズレと一緒です。しかし、ここまでハズレが連続したら、当たる時は一等に違いないはず。それが、唯一の心の支えです。やっぱり私は楽天的でしょうか。

でも、最近は子供も大きくなってきたし、少しはシビアになろうと思い、宝くじの番号を指定して購入するようにしました。もちろん、いつも思い通りの番号が買えるとは限りません。しかし、少なくとも運は向上すると思うのです。ただ、今のところ結果は同じでハズレばかりです。

そこで、今度は購入番号をパソコンの乱数に選ばせようと思います。乱数で各桁ごとの番号を決めたいのですが、私の知っている乱数（５倍して１を足す）では、すぐループしてしまい面白味がありません。

もう少し変化に富んだ乱数はできないでしょうか。

宝塚九二男（兵庫）

答

宝くじ。本当に当ててみたいものです。よく「運がよければ……」といいますが、いったいどの程度の運なのでしょうか。

一等の出る確率を計算すると、宝くじによっても、また一等賞金の額によっても違いますが、だいたい百万分の一とか、二百万分の一といった感じが多いようです。いずれにしても、桁が多すぎてピンと来ない数字です。

では、直径１ミリの砂つぶをすき間なく1,000メートル並べ、その中の１つぶだけが一等になるといったらどうでしょう。……書いてみて、確率の低さに驚いたのは、他ならぬ自分でした。

さて、例の５倍して１を足すという乱数（問その30）ですが、コールするたびに偶数／奇数が入れ替わり、下位４ビットは16回で一巡してしまいます。全体的にも 256回で元の値にもどりますから、利用できるのはせいぜい簡単なゲーム程度と考えたほうがよさそうです。

そこで、宝くじに応用できるような複雑な乱数の登場です。といっても、基本的な考え方は５倍して１を足す方法と同じです。ただ、計算式を16ビットで

行い、その上位8ビットを乱数に取るので、その値からは次回の乱数を予測できなくなります。

```
RND2 :  LD      HL,0
        LD      D,H
        LD      E,L
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,DE
        LD      DE,3711H
        ADD     HL,DE
        LD      (RND2+1),HL
        LD      A,H
        RET
```

最後に加える 3711_H は特に意味のない数値であり、0000_H 以外であればいくつでも構いません。しかし、数値の変化を早めるためには、少なくとも 0100_H 以上は入れたほうがいいでしょう。また、この値により乱数に連続性などの性格を付けることができます。色々な値を入れてテストしてみてください。

ただし、単純計算による乱数は必ずループします。また、一巡する間に出る数値は、順序が変わってもすべて同じ回数です。例えば、3711_H の代わりに 0001_H とすると、0 が連続することが多くなりますが、最終的には他の数と同じ割合でしか現れません。

ちなみに、この乱数は一巡するのに 10000_H (65536) 回かかります。したがって、次の値や偶数／奇数を予測することは、計算しない限りまず不可能です。参考までに、今回のプログラムによる最初の 100_H (256) 回の乱数を示しておきます（次頁）。

もし、当たりそうな乱数が得られるようだったら、ぜひ教えてもらいたいものです。これは本気です……。

```
37 4A AB 8E FE 31 2D 1C C7 1E D0 47 9E 4E BD EA
CD 3B 62 22 E2 A3 6A 49 A8 82 C1 FD 2B 12 93 19
B8 D1 4D BA DA 7A 9A 3A 5D 09 66 37 4D BA DD 8C
F7 0A 6C 56 E5 B4 BE EF E5 B4 BF F5 03 47 9C 43
89 E8 BF F6 05 53 D6 68 42 84 CC 37 4C B7 CE 3E
70 69 46 9A 39 55 E2 A5 73 77 8E FD 2A 0C 74 7D
AB 8F 02 41 80 BC E3 A6 78 8F 03 47 9C 44 8E 00
39 58 F1 ED DC 86 D7 6B 50 CA 2C 15 A2 61 1C C7
1C C5 14 9D 4C B4 BF F4 FD 2A 09 67 3B 61 1F D2
53 D7 6B 51 D0 47 9B 41 7E AD 9A 3D 69 46 95 21
DD 8C F6 09 67 3D 6B 52 D2 54 E0 97 2B 0E 7F B4
BC E6 B6 C5 13 98 30 27 FB 20 D9 75 80 BA DD 8B
EF E3 A9 85 D3 56 E8 C0 F8 0F 86 D7 6A 4B AF A6
76 85 D0 49 A6 79 94 1D C8 23 E7 BD E8 BF F6 05
50 CA 2B 11 8E FF 34 3E 6D 5A FC 26 F9 18 B0 A8
7F B3 BA DD 8A E9 C8 23 E6 B6 C5 14 9F 54 DE 8F
```



問その32 疑似乱数利用の基礎

おいでませー、山口へ。山口といえば長州。長州といえば、維新の嵐。革命戦士の心意気でマシン語を制覇するゾ!!

……と意気込んでみたけど、いま挫折しかかっています。ぼくはプロレスラーと精神構造が一体化している高１です。小さい時からバーベルで鍛えてきただけあって自慢の胸囲は130cm を誇ります。

挫折の原因は単純です。なにかにつけて乱数乱数といいますが、ぼくには乱数の意味はわかっても、それをどう利用するかがわからないのです。例えば、ゲームなどでは敵の動きを乱数で決めることがあるそうですが、動きは８方向程度なのに、乱数では 0〜255 までの数ができてしまいます。

例えば、1〜8の乱数がほしければ、その数が出るまで乱数ルーチンをコールしろということでしょうか。

```
              :
CRND1 :  CALL   RND1          ←「RND1」は問（その30）
                                の乱数ルーチン
         CP     8
         JR     NC,CRND1
         INC    A             ← A＝1〜8の乱数
              :
```

あるいは、もっと特殊な乱数ルーチンがあるのですか。どうか、この挫折から立ち直れるよう、カツを入れてください。

男一匹長州軍団（山口）

答

喝（かつ）!!

これで、挫折から立ち直れるなら何度でもカツを入れてあげましょう。喝!!

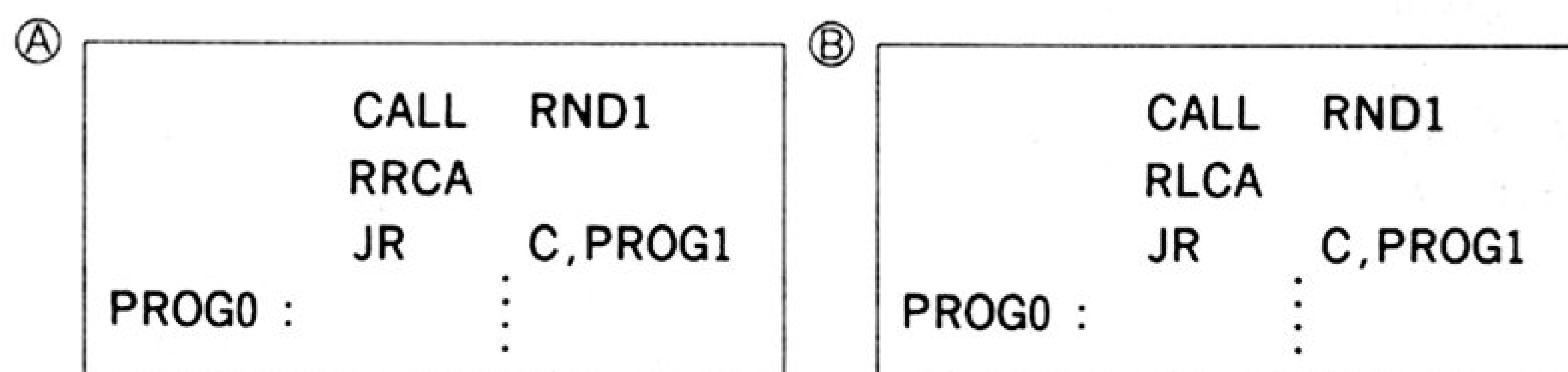
なに、まだ挫折から立ち直れない……。喝!!

エーッ、まだ……。よく考えたら、こんなことをするより、先へ進んだほうが早く立ち直れそうですね。では、乱数の利用法へと進むことにしましょう。

乱数は、作るよりどう利用するかでその価値が決まります。料理の味がコックさんの腕前で変わるように、乱数もうまく活用することが大切なのです。どんなにいい乱数ルーチンでも、利用方法が悪ければ結果的に悪い乱数になって

しまいます。

例えば、5倍して1を足すという簡単な乱数（問その30）を用い、確率 1/2 の割合で条件分岐させるとします。



Ⓐ
```
          CALL    RND1
          RRCA
          JR      C,PROG1
PROG0 :            :
```

Ⓑ
```
          CALL    RND1
          RLCA
          JR      C,PROG1
PROG0 :            :
```

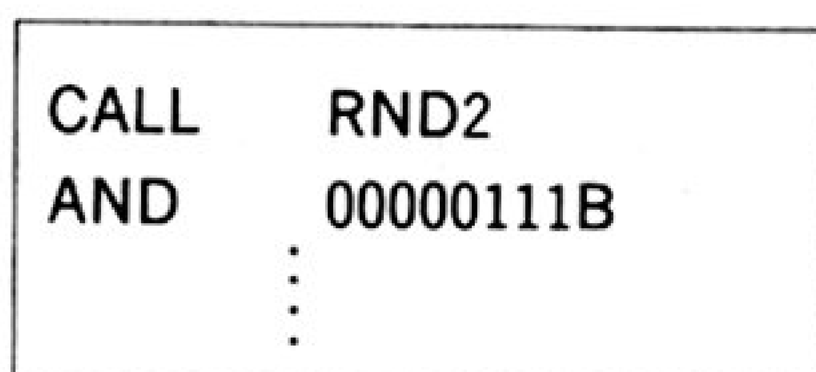
ⒶもⒷもプログラム的には似たようなものです。しかし、得られる結果はまったく違います。Ⓐのほうは、キチンと1回おきに分かれてしまい、乱数を取った意味がなくなっています。一方、Ⓑのほうはある程度予測不可能な分岐をしてくれます。

もちろん、どちらも 256回実行すれば分岐割合は同じですが、乱数ルーチンは何もこのプログラムだけで利用されているとは限りません。下手をするとⒶのプログラムは同じ場所へのジャンプしかしなくなる可能性だってあるわけです。

したがって、簡単な乱数を利用する時には、その乱数の性格も頭に入れて利用することが大切です。「RND2」(問その31)を利用すればそういう心配は不要ですが、「RND1」には「RND2」よりプログラムが短い上、速度も速いという特徴があるのです。一概に役に立たないというのではなく、場合による使い分けをしたいものです。

では、質問にあるような数値を乱数で作ってみましょう。乱数ルーチンは「RND2」を使用するものとします。これは、「RND1」では下位4ビットが16回でループしてしまい、今回の応用には適さないからです。

(1) 乱数から 0〜7 の数を作る



```
CALL    RND2
AND     00000111B
          :
```

⑵ 乱数で 1〜8 の数を作る

```
CALL    RND2
AND     00000111B
INC     A
         :
         :
```

要するに、乱数ルーチンで得た数値をそのままの姿で利用するか、それとも加工して好みの数値にするか、それを決めるのは「あなた」というわけです。知ってしまえば簡単なものですね。

挫折から復活の呪文へ……喝（かつ）!!



問 その33

疑似乱数の応用

わたくし生まれも育ちも葛飾柴又です。

帝釈天で生湯をつかり、姓は車、名は寅三郎。

人呼んで『フーセンの寅』と発します。以後よろしうおたの申します。

わたくしの職業は、全国津々浦々に出向き、立地条件のよい場所を見つけては風船の直営店を開くことであります。そのためには、全国の祭り、縁日、運動会、バザーにイベント、歩行者天国。あらゆる催し物の場所と日時を、正確に記憶しておくことが大切なのでございます。

カバンひとつで長期の出張、つらいけれども気楽な身です。唯一の不安は、行き先を間違えて商売をし損なうことです。しかし、いまでは一枚のフロッピィディスクがあるので安心です。そこには、柴又のオイちゃんの家でインプットした、全国の催し物の情報が入っております。こう見えても、プログラムは自前、それもマシン語です。

お祭り好きな日本では、いつもどこかで祭りが開かれています。客足の予想なども、そのプログラムで知ることができます。しかし、欠点はいつも同じ結果しか表示してくれないことです。

そこで、乱数を利用して上下に変化をつけたいと思うのですが、乱数をどのように活用したらいいものか、悩んでおります。どうかよろしくご教授くださいませ。

ちなみに、プログラムはデパートで実行させております。

旅先にて……風船のトラ（福井）

答

地方を巡業中のフーセンの寅さんも、コンピュータで行動を管理していたのですか。気ままな商売に見えても、結構苦労していたんですねェ。

どのようなプログラムで客足の予想までしているのか、これだけではわからないのが残念ですが、とりあえず、＋／－を含んだ乱数値を得るのが目的のようです。もっとも、問（その1）にあるように、＋／－というのは使う側の勝手ですから、何もしなくてもすでに －128～＋127 の数値として扱うことはできるわけです。

しかし、これでは上下の幅が広すぎますね。計算をしたとたんに8ビットの限界を超えて、足したつもりが少なくなってしまう恐れもあります。

(例)

150（元の数値）＋120（変動幅）＝ 14 ← 8ビットの限界を超えたため

中には、客足の予想をするのに、8ビットではそもそも無理があると感じる人もいるでしょう。しかし、数値には単位というものがあります。例えば、100という値でも後ろに「00」を付ければ 10000を表現したことになるように、単位次第でいくらでも表現できる幅はふくらむのです。

ゲームのスコアなど、内部では1点単位、画面では飾りの「00」を付けて 100点。そんな見え見えのゴマカシもあるほどです。ということで、「RND2」(問その31) を利用して、上下幅のある数値を作ってみます。

(1) −7〜+7の数を作る

```
        CALL    RND2
        AND     00001111B      ←「AND  10000111B」としただけでは
                                  −7〜+7 とはならないことに注意
        RRCA
        JR      NC,NONEG
        NEG
NONEG :          :
```

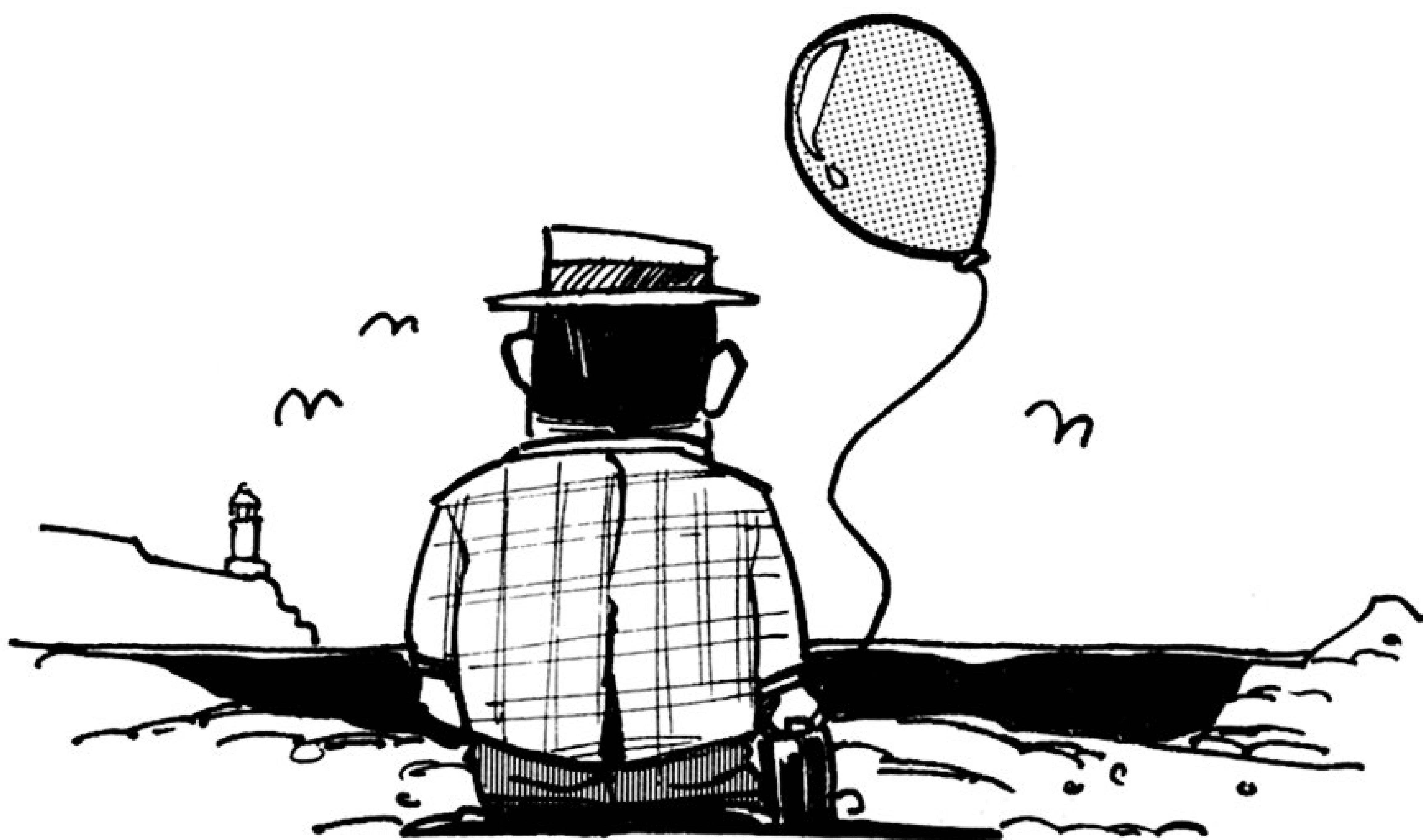
(2) 30%の確率で雨が降る。雨ならば−7〜−4の数値を、雨でなければ+4〜+7の数値を作る (256/100=1%と考える)

```
        CALL    RND2
        AND     00000011B
        OR      00000100B      ←「ADD  A,4」でもよい
        LD      B,A            ←B=4〜7の乱数
        CALL    RND2
        CP      77             ←30%の確率 (77=30×256/100)
        LD      A,B
        JR      NC,NRAIN
        NEG
NRAIN :          :
```

これらの例のようにAND命令がいつもうまく使えるとは限りませんが、加減算をして調整したり、確率による条件分岐を組み合わせることで、たいていは条件に見合った乱数値を作り出すことができます。

どうやっても条件からはみ出してしまう数値が出る場合は、条件に見合うまで同じ処理を繰り返すか、確率の狂いは無視して適当な数値に決めてしまうことです。いい加減でいいのが乱数なのです。

タコはイボイボ、ニワトリゃはだし。イモムシゃ19で嫁に行く。乱数いろいろ、よく見りゃ数字。足して引いてりゃ丸くなる……。



問その34

疑似乱数に変化をつける

わたしは女の子ですが、パソコンが大好きです。とくに、面白いゲームがどういうふうにプログラムされているかを考えると、胸がワクワクします。

ゲームとしては、アクションゲームよりか、ドラクエみたいなRPGが大好きで、わたしもいつの日かあんなゲームを作ってみたいナと思います。だから、少しはマシン語も勉強しました。

敵とか出すのって、乱数を使うんでしょ。それくらいは知っています。でも、ゲームでいう乱数は疑似乱数のはずです。だから、ゲームをスタートして同じように動けば、誰がやっても同じ敵が同じ場所に出てくるはずだと思います。

ところが、ためしにそういう実験を友人とやってみると、ソックリ同じにはなりませんでした。もしかすると、本物の乱数ですか……？

中3あきな（石川）

答

女の子のパソコンファンが近ごろ増えてきているらしいですね。これで、パソコンの未来も明るくなりそうです。

パソコンの魅力の1つに、市販ソフトとまったく同じものを本体で作ることができるという特徴があります。これは、プログラムを組まない人にとっては無用の長物です。その点、遊ぶ面白さから作る面白さに気が付いた石川のあきなちゃんはエライっ!!

未知のものを作る時のスリル、そしてプログラムを初めて実行する時の不安感。これはもう体感RPGそのものです。プラモデルだって、完成してからより、作っている時のほうが楽しいし夢があります。

TVゲームで遊んでいる子供たちが、そういったパソコンの特徴に気が付いた時、パソコンは飛躍的に広がるでしょう。

さて、この疑似乱数に関する疑問は非常に鋭い点を突いています。おまけに実験までしているとは、おそれ入谷の鬼子母神です。おっと、トラさんからの質問があったばかりなので、つい口がすべってしまいました。

しかし、このくらいのことはプログラマーならば対処して当然のことなのです。というのは、乱数の初期値を変えるだけで、問題はアッサリと解決してし

まうからです。

では、どのようにして乱数の初期値を変えたらいいでしょうか。乱数の初期値に乱数を使うというのでは「卵が先かニワトリが先か」になってしまいます。悩みそうな問題ですが、いくつかの簡単な解決法があります。

(1) 名前の合計（文字もマシン語上では数値です）を乱数の初期値にする

これはRPGなどではよくやりますが、名前が同じだと合計も同じになってしまいます。しかし、それも名前による一種のゲーム性です。このプログラムでは、文字コードをアスキーコードで表現し、0 を名前のエンドサインにしています。

```
NAMAE :  DB      'アキナ',0
START :  LD      HL,NAMAE
         LD      DE,0
STAT1 :  LD      A,(HL)
         OR      A
         JR      Z,STAT3
         ADD     A,E
         JR      NC,STAT2
         INC     D
STAT2 :  LD      E,A
         INC     HL
         JR      STAT1
STAT3 :  LD      (RND2+1),DE
            :
```

(2) キー入力のループの中で、乱数ルーチンをコールする

一般に、どんなゲームでも最初にタイトルなどがあり、なにかキーが押されるのを待つようになっています。そのキー入力チェックのループで、乱数ルーチンをコールすれば、ゲーム開始時には乱数にズレが生じているというわけです。

(3) R（リフレッシュ）レジスタの値を乱数の初期値とする

パソコンに使用されているRAMはダイナミックRAMといい、短時間でメモリの内容が消えてしまう性格を持っています。そのため、内容が消

える前に再びメモリに書き込むという作業を常に行っているのです。そのカウンタとなっているのが、Rレジスタです。
リフレッシュは一度に 512バイトずつ実行されるので、Rレジスタは7ビットで全メモリ空間をカバーしています。これは、内部で自動的に行われますから、Rレジスタの値は常に変化しています。ただし、+1ずつ変化していくので、このまま乱数として使用することは不適当です。

```
START:   LD    A,R
         LD    (RND2+1),A
```

これだけで、乱数初期値の下位7ビットが不定になります。シンプル イズ ベストとするなら、これがベストといえるでしょう。

あきなちゃんだけでなく、せいこちゃん、きょうこちゃん、のりこちゃん。それに、さゆりちゃん、わかこちゃん……。みんなパソコンファンになればいいのに……ネ。



問 その35 裏レジスタとは

あなたは神を信じますか？

わたしは神を信じます。わたしはニューギニアにいるドイツ人の牧師です。わたしは神の教えを広め、人々の幸せと平和を祈り、貧しい者、富める者、病める者、健康な者、すべての人々の心の悩みを救うため、ここニューギニアへやって来ました。

ところで、わたしの友人に変な日本人がいます。彼は、わたしに悩みを解決してくれと言います。ところが、その悩みとはなんとマシン語についてです。これには、わたしも困ってしまいます。

彼の話によると、ある日裏レジスタの存在を知り、これは便利と PUSH/POP の代わりに使用したそうです。最初のうちは調子が良かったのですが、そのうち表か裏かワケがわからなくなり、プログラムがメチャクチャになったというのです。

彼は、そのことで悩み、わたしに相談してきました。わたしは人の悩みを解決するのが仕事です。わたしはこの手紙を英語で書き、それを彼に日本語に訳してもらいました。

だから、この手紙はわたしの手紙であっても、書いたのは彼であり、悩んでいるのはわたしであっても、悩みは彼のものであり……。なんで、わたしが悩むのか、それがまた悩みとなって……ナニがどうなっているのか、わからなくなりました。

オー、神よ。悩めるわたしを救いたまえ。アーメン……。

悩める牧師（ニューギニア）

答

「あなたは神を信じますか？」

「信じます。ただし、困ったときだけ」

……というのが、多くの日本人だそうです。無理もないですネ。生まれた時は神社にお宮参り、結婚式は教会で、お葬式はお寺で、そして普段は祈らない……。これが通用するのが日本ですから。

そんないい加減な信者の一人ですが、いつものお礼に悩める牧師さんを救ってあげたいと思います。

まず、裏レジスタという名称ですが、一応区別するために表／裏、あるいは主／副レジスタと言っていますが、実際には同じ土俵で使えないというだけで、表とか裏という感覚のものではありません。

というのは、裏レジスタに対する命令というのは存在しないし、使用する際には次のように表と裏を交換して使うからです。つまり、交換した時点で、裏

が表に、表は裏にと立場が逆転してしまうのです。

表レジスタ	交換命令	裏レジスタ
AF	EX AF,AF'	AF'
HL,DE,BC	EXX	HL',DE',BC'

この特徴を利用して、PUSH/POPの代わりに利用できないことはありません。特に、一度に3つのペアレジスタを交換できるEXX命令は、PUSH/POPに比べ記述が楽な上、メモリの節約、速度のアップにもつながっています。ついつい PUSH/POP の代用で使用したくなりがちです。

しかし、こういった使い方は、メイン側を表レジスタだけに限定し、割込み処理ルーチンでレジスタ退避の代わりに裏レジスタを使用する、というように明確な環境設定をした場合でないと非常に危険です。

というのは、PUSH/POP代わりに使用すると、ジャンプ先やコール先でEXX命令を勝手に使用することもあり、何度も繰り返しているうちに、どっちがどっちかわからなくなってしまうからです。

したがって、裏レジスタをメイン側で使用する場合は、表レジスタの退避として使うのではなく、キチンとした役割を持たせて使うべきです。例えば、次の不規則転送プログラムはデータを16バイトごとに読みだし、別のアドレスに8バイトごとにストアするというものです。このような場合、裏レジスタが非常に便利な存在となります。

```
        LD      HL,0C000H       ←転送データのアドレス
        LD      DE,16           ←スキップ数
        LD      B,20            ←総転送数
        EXX                     (裏)
        LD      HL,0E000H       ←転送先アドレス
        LD      DE,8            ←スキップ数
        EXX                     (表)
LOOP :  LD      A,(HL)
        ADD     HL,DE
```

```
        EXX                 (裏)
        LD      (HL),A
        ADD     HL,DE
        EXX                 (表)
        DJNZ    LOOP
        RET
```

要するに、表は表、裏は裏。それぞれに目的と役割を持たせて使えばいいのです。そのためには、裏レジスタを表レジスタの退避用と考えるのではなく、独立した別のレジスタとして存在価値を認めることがポイントです。

これで、悩める牧師さんを救えたとしたら、わたしは神となるのでしょうか。オヤ、本当の神の声が聞こえてきました。

あなたは神ではなく紙。つまり本です、アーメン……。



問 その36 表／裏レジスタの共用

ワシは悪魔神官バーボン……。

つまり、酒の好きなおまえたち人間の酔った時の心を司る神じゃ。酒は人の心を気持ちよく狂わせ、理性を失わせる。酒は百薬の長、祝い酒、おとそ、おみき……、飲む理由はいくらでもある。

そこがワシのつけ目じゃ。やがて、ジキルとハイドのように、まったく違った人間ができ上がる。それが、おまえたちの裏の心だ。わかるか、心に表も裏もない。裏が表になれば、それが表になるのだ。その時こそ、世界がワシのものになる。

……というのが、ワシが描く理想の世界なのだが、どうも理想と現実は違うような気がする。いつまでたっても、理想の世界にならんのじゃ。

そこで、ワシは人の心をコンピュータで分析しようと思う。うまい具合いにレジスタにも表と裏があるではないか。完全に酔ってしまえば、全部のレジスタを交換してもいいのだが、最初のうちは裏レジスタと表レジスタに共通部分が必要なのだ。

つまり、表で使用していたレジスタの値を、そのまま裏レジスタへ移すにはどうしたらいいか、というのが質問じゃ。分析が終わったからといって、ワシの理想通りになるとはかぎらんから、安心して質問に答えてくれ。

天才バーボン（酒乱界）

答

これは、恐ろしい質問です。私が酒飲みなら、不安でとても答えられないのですが、幸か不幸かわたしは酒が飲めません。だから、平気で質問に答えることができます。

例えば、表にあるHLレジスタの値を、裏のHLレジスタに移したいなら、次のようにすれば簡単です。

```
PUSH    HL
EXX
POP     HL
          :
```

この方法で、何度でも表と裏を共通して使用することができます。もちろん、裏から表へ移す時も同じです。また、インデックス・レジスタ（IX／IY）は表も裏もありませんから、共通のペアレジスタとして常に利用可能です。

(例)1000_H 番地からのデータを80バイトおきに読みだし、3000_H番地から連続してストアする。同様に、2000_H番地からのデータを40バイトおきに読みだし、3020_H番地から連続してストアする。この作業を 20_H (32) 回実行する。

```
        LD      HL,1000H
        LD      DE,80        ←スキップバイト数
        EXX
        LD      HL,2000H
        LD      DE,40        ←スキップバイト数
        EXX
        LD      IX,3000H     ←転送先を示す共用
        LD      B,20H          のレジスタ・IX
LOOP:   LD      A,(HL)
        LD      (IX+0),A
        ADD     HL,DE
        EXX
        LD      A,(HL)
        LD      (IX+20H),A
        ADD     HL,DE
        EXX
        INC     IX
        DJNZ    LOOP
                :
```

では、表のBレジスタだけを裏のBレジスタへ移す場合はどうでしょう。PUSH/POP で移したのでは、Cレジスタの値も移ってしまいます。

こういう時は、表と裏に共通して使えるAレジスタを利用します。

```
LD      A,B
EXX
LD      B,A
        :
```

裏と表は、うまく利用すれば倍の能力を発揮してくれます。酒を飲むと頭がさえるという人間もいることを、お忘れなく……悪魔神官バーボン殿。

問 その37 メモリを埋める

佐渡へ一佐渡へーと、草木もなびィーく〜……。

と、歌われたのも今は昔のお話です。最近では、佐渡で金が取れたことさえ知らない人がいます。しかし、私はまだまだ佐渡で金は出ると信じています。だからこそ、全財産を処分してここへやって来たのです。

私は、自称地質学者。他称『ヘンなおじさん』です。マ、多少は他称のほうが当たっているかもしれませんが、そんなことはどうでもいいことです。現在、コンピュータにより地層を分析し、夢にまで見た金脈を探しています。

もっとも、コンピュータといっても中古のパソコンです。独学で覚えたマシン語を使っていますが、実行速度が遅く、なかなか分析は進みません。ここにあるのはメモリを一定のデータ（ここでは 0）で埋めるルーチンです。

```
          XOR     A
          LD      HL,xxxx
          LD      B,120
FLOOP :   LD      (HL),A
          INC     HL
          DJNZ    FLOOP
          RET
```

何度も使われるので早ければ早いほどいいのです。もし、うまく改善してくれたら、金が出た時に 1 kg ほどあげましょう。

名物・金脈おじさん（佐渡島）

答

1 kgの金……ということは、時価に換算すると……ウ〜ン。いずれにしても大金であることは間違いありません。なんとしても、この質問にだけは答えなければ……。

まず、質問にあるプログラムが何ステートかかっているか、それを計算してみます。その後でどの程度のスピードアップが計れたか、改善したプログラムと実際に比較してみましょう。

```
 4 ×   1 =    4 ··· XOR    A
10 ×   1 =   10 ··· LD     HL,xxxx
 7 ×   1 =    7 ··· LD     B,120
 7 × 120 =  840 ··· LD     (HL),A
 6 × 120 =  720 ··· INC    HL
13 × 119 = 1547 ··· DJNZ   FLOOP（ループ時）
 8 ×   1 =    8 ··· DJNZ   FLOOP（通過時）
─────────────────
 TOTAL :  3136ステート
```

改善案－1

```
XOR     A                 4
LD      HL,xxxx          10
LD      DE,xxxx+1        10
LD      BC,119           10
LD      (HL),A            7
LDIR                     21 × 118 + 16 = 2494
RET
                         ─────────────────
                         TOTAL : 2535 ステート
```

1バイトずつメモリを埋めていたのを、転送命令に置き換えただけですが、実行速度は約2割ほどアップしています。これはかなり実用性の高いやり方ですが、速度だけを追求するならもっと早くなります。

改善案－2

```
         DI                          4
         LD     (SPPOP+1),SP        20
         LD     SP,xxxx+120         10
         LD     HL,0                10
         LD     B,12                 7
SLOOP :  PUSH   HL                  11 × 12 = 132
         PUSH   HL                  11 × 12 = 132
         PUSH   HL                  11 × 12 = 132
         PUSH   HL                  11 × 12 = 132
         PUSH   HL                  11 × 12 = 132
         DJNZ   SLOOP               13 × 11 + 8 = 151
```

```
SPPOP :  LD      SP,0       10
         EI                  4
         RET
                           ─────────
                           TOTAL : 866ステート
```

改善案－2は、SP(スタックポインタ)の特殊な使用例で、PUSH命令でデータが2バイトずつメモリにストアされる特徴を利用したものです。ステート数を見ると、2バイトずつ処理された効果が明確に出ています。

ここで注意しなければならないのは、この間は本来のSPの働きができないということです。つまり、通常のPUSH/POPによるレジスタの退避はもとより、サブルーチンをコールしたり、割込みをかけることもできません。したがって、いつでも利用できるとは言えませんが、速度的な魅力はかなりのものがあります。

さらに、メモリにゆとりがあるなら、「PUSH　HL」を60回連続して記述すればループに消費していた分（7＋151＝158 ステート）が不要になり、トータルで 708ステートとオリジナルの 1/4以下にすることができます。

これで、金1kgが私のものに……なるわけないでしょうね。



問 その38 データ転送

わが輩は、あの有名な'ねずみ小僧次郎吉'の子孫にあたる者だ。もっとも、おまえらだってその子孫の一人かもしれないがな。なにしろ、ご先祖様の子孫は全国にネズミ算式に増えているから、今では誰が子孫かわからなくなってしまったのだ。

だが、わが輩こそ本家本元家元元祖、ただ一人'ねずみ小僧'を名乗れる資格のある子孫なのだ。なぜかって？　映画で見たご先祖様のご尊顔とソックリだからだ。どうだ、まいっただろう。

ご先祖様は、お金をアルところからナイところへ移動していたが、わが輩が移動するのはお金ではない。メモリにあるデータを、アルところからナイところへ移動しようというのだ。わが輩のプログラムを見てくれ。

```
         LD      B,10
MOVEL :  LD      A,(HL)
         LD      (DE),A
         INC     HL
         INC     DE
         DJNZ    MOVEL
                  :
```

ご先祖様は、日本一のスピードでお金を移動した。わが輩のプログラムは、その子孫として恥ずかしくないものかどうか、コソッと教えてもらいたい。

根津見狐造（住所不定）

答

ねずみはネズミ算式に増えるといいますが、昔も今もねずみの数は余り変わらないような気がします。もし、大昔から計算通りに増えていたら、今ごろは世界中がねずみだらけになっていたはずですが、いったいどうなってしまったのでしょう。実に不思議なネズミ算の実体です。

さて、このプログラムではちょっとばかりご先祖様には恥ずかしいようです。ループ命令まで入れると、1バイト転送するのに39ステートもかかっている上、プログラムには速度を追求した形跡がまったくありません。せめて、次のようにするべきです。

```
LD      BC,10
LDIR
```

これならば、1バイトにつき21ステート（最後の1バイトは16ステート）です。約半分になっていますが、思い切ってメモリを消費すれば、さらに高速化できます。

```
LDI    ←16ステート
LDI
LDI
LDI
LDI
LDI
LDI
LDI
LDI
LDI
```

このように、ループを省いてしまうというのは高速化の常とう手段です。「LDIR」そのものは1つの命令ですが、内容はループを含んだ転送命令ですから、ループになる時は余分に時間がかかってしまうのです。したがって、本当に高速化が必要な場合には、この命令を平然と 50個でも 100個でも並べることがあります。

また、「LDI」や「LDD」を連続させる場合、最初に「LD　C,0FFH」としておくと 255回の連続まではBレジスタの値が変化しないので、Bレジスタを別のカウンタなどに利用することが可能です。「LDIR」や「LDDR」命令では、BCレジスタが 0になるまでループしてしまうので、Bレジスタを他の目的に使用することはできません。こういった特徴も覚えておく価値があります。

では、ご先祖様に負けないよう頑張ってください。

問その39 データ高速転送

わたしは女の子よ。でも、ちょっと普通の子じゃないの。あまり大きな声じゃ言えないけど、ド・ロ・ボ・ウなの。ア、誤解しないでね。悪人じゃないんだから。

わたしのこと、世間では「怪盗ルビィの指輪」って呼んでいるみたいだけど、普通の人は知らないと思うわ。だって、それは泥棒さんたちの呼び名ですもん。

わたしのお仕事は、泥棒から盗まれたモノを盗り返して持ち主にこっそり返すこと。だから、泥棒さんたちから見れば、わたしが泥棒になるの。もちろん、相手が相手だけにお仕事は楽じゃないわよ。なにしろ相手もプロだから手も足も結構すばやいの。だから、それに負けたらオシマイね。

一番必要なのが頭のすばやさ。パソコンでプログラムを組むときも、高速化は常に考えているわ。データの移動に「LDI」や「LDD」を連続して書くのなんか当然よね。

でも、あの命令ってどちらにしても一方通行でしょ。つまり、HL が＋なら DE も＋、HL が－なら DE も－。わたしがしたいのは、HL が＋で DE が－という転送なの。ここにあるのじゃ平凡よね。

```
        LD      HL,DATA1
        LD      DE,RUBI1
        LD      B,20
XLOOP : LD      A,(HL)
        LD      (DE),A
        INC     HL
        DEC     DE
        DJNZ    XLOOP
        RET
```

やっぱりこれしかないかしら。でも、こんなんじゃ泥棒さんのレベルと一緒よね。きっと、なにかいい方法がありそうな気がするんだけど……。教えて!!

怪盗ルビィの指輪ちゃん（広島）

答

なんか、こうフラフラッと教えたくなるようなお手紙です。きっと、相手の泥棒もすばやさを忘れて、ついボーッとしている間に盗り返されてしまうんでは……。

しかし、プログラムにまで高速性を追求しているとは、まさに本物のプロといった感じです。でも、本当に女の子なんでしょうか。もしかすると、これも敵を欺くための手段かもしれません。

さて、このプログラムを高速化するには、ちょっと頭をヒネらなければいけません。というのは、常識的に考えればこれ以外にプログラムを組む方法がないからです。というわけで、頭をヒネった結果が次のようなプログラムです。

```
        DI
        LD      (SPPOP+1),SP
        LD      SP,DATA1
        LD      HL,RUBII
        LD      B,10
XLOOP:  POP     DE              ←10ステート
        LD      (HL),E          ← 7ステート
        DEC     HL              ← 6ステート
        LD      (HL),D          ← 7ステート
        DEC     HL              ← 6ステート
        DJNZ    XLOOP
SPPOP:  LD      SP,0
        EI
        RET
```

プログラムの特徴としては、POP 命令でデータを一度に2バイトずつ読みだしていることです。そのため、2バイトを転送するのに36ステートで済んでいます。質問にあった方法では、ループせずに2バイトを転送したとしても52（26×2）ステートかかります。

つまり、約3割のスピードアップが実現したわけです。一方、この間はSPを転送するデータアドレスとして使用していますから、当然のことながらレジスタを退避したりサブルーチンをコールしたりすることはできません。もちろん、割込みも禁止しておかなければなりません。

言ってみれば、これが高速化の代償というわけですが、データが多くなればなるほどこの効果は絶大なものとなるでしょう。また、この転送部分をループを使わずに連記すれば、ループによる無駄が省けさらに高速になります。

せっかくの高速化テクニックですが、こうして本になると泥棒さんにも見られてしまうのでは……？

問 その40 3バイトの加減算

メンソーレ。ぼくはパソコンが趣味の剣道部員です。得意技は、正眼からの鋭いメンです。かけ声は、もちろん「ソーレ!!」です。

ところで、16ビットのレジスタでは0〜65535(0〜$FFFF_H$)までしか計算できません。いまどき、子供の貯金だってそれ以上です。ただし、ぼくの場合はパソコンを買ったばかりなので、それ以下ですけど……。

でも、ロールプレイングゲームなんかをすると、経験値やゴールドはどんどん上がっていきます。もちろん、どこかに上限はあるでしょうけど、ぼくにはどうやっているのかわかりません。

もしかすると、秘密のレジスタでもあるのでしょうか。教えてくれたら、お礼に元気が出る'ハブのエキス'を差し上げましょう。

青い珊瑚礁（沖縄）

答

青い海、澄んだ空、まぶしく輝く太陽。そんな沖縄で見た「ハブとマングースの決闘」は忘れられません。あの時、一匹目のマングースは無残にもハブの毒牙にやられてしまったのです。勝っても負けても、勝負は一瞬で終わりでした……。

さて、問題の16ビット以上の計算方法についてですが、もちろん秘密のレジスタなどあるわけがありません。レジスタがダメなら、頼れるのはメモリだけです。メモリなら何バイト使おうと自由ですからね。例えば、メモリを3バイト使えば、表現できる限界は一挙に 0〜16777215と大幅アップします。

ただし、当然のことながら3バイトを一挙に加減算できる命令などありません。したがって、すべてプログラムで用意することになります。では、3バイト使用した数値どうしの加算と減算のプログラムを組んでみましょう。それぞれの数値はアドレスの若いほうを上位桁とし、計算結果はメモリ（KEKKAからの3バイト）に入ります。

DATA0 + DATA1

```
DATA0 :   DB      1FH,20H,5AH
DATA1 :   DB      13H,50H,88H
```

DATA0 − DATA1

```
DATA0 :   DB      1FH,20H,5AH
DATA1 :   DB      13H,50H,88H
```

```
KEKKA :  DB    0,0,0
ADD01 :  LD    DE,DATA0+2
         LD    HL,DATA1+2
         LD    IX,KEKKA+2
         OR    A
         LD    B,3
ADDLP :  LD    A,(DE)
         ADC   A,(HL)
         LD    (IX+0),A
         DEC   DE
         DEC   HL
         DEC   IX
         DJNZ  ADDLP
         RET
```

```
KEKKA :  DB    0,0,0
SUB01 :  LD    DE,DATA0+2
         LD    HL,DATA1+2
         LD    IX,KEKKA+2
         OR    A
         LD    B,3              ←B
SUBLP :  LD    A,(DE)           =桁数
         SBC   A,(HL)
         LD    (IX+0),A
         DEC   DE
         DEC   HL
         DEC   IX
         DJNZ  SUBLP
         RET
```

どちらのプログラムにおいても、重要なキーポイントになっているのはキャリーフラグです。また、16ビットのDEC命令やDJNZ命令ではフラグが変化しないという特徴が、このプログラムを成立させています。

これらのプログラムを実用化する場合、HL／DEレジスタを引数としたサブルーチンとすれば、各種の3バイトデータを計算させることができます。また、減算ルーチンはCP命令としての機能も兼ね備えていますから、このルーチンをコールしたあとで、ゼロフラグやキャリーフラグによる条件分岐ができます。

桁数はメモリの許す限り（Bレジスタのカウントできる範囲ですが）増やすことが可能です。もっとも、それを利用する人間が何桁もの十六進数を数値として理解できるかどうかは別問題ですが……。

問その41

十六進数を十進数に

十数年前、私は海外で番を張っていました。……というと、おそろしい話に聞こえるでしょう。でも、これは本当の話です。

私は、ケンカも弱く嫌いです。でも、旅行は大好きです。ユースホステルでは、よく何連泊もしたものです。

ある時、連泊が1ケ月以上になりました。すると、いつの間にか副番長の肩書きが付きました。やがて番長が日本へもどることになり、なんと私が番長へ昇格したのです。そこでは、日本人の最長連泊者を番長とする習わしだったのです。

いま、いい年してパソコンに凝っています。マシン語でも 0〜9 の数字を画面に表示することができるようになりました。それは、Aレジスタに入れた 0〜9 の値を画面に表示するという単純なルーチンです。

しかし、マシン語上で扱う数値は十六進法です。これを十進法に変換し、一桁ずつAレジスタに入れられなければ、私のルーチンは役に立ちません。

どうするべきでしょうか。それができないことには、仕事も手につきません。すでに自室へ15連泊もしています。もう、番長は結構だ……!!

番長ボロ屋敷（宮城）

答

知っています、その話。スペインのマドリッドのユースでしょ。あそこでは日本人の番長が代々「火縄式ライター」を安く売ってくれるというので、私もノコノコと出かけて行ったことがあります。大きな日の丸を掲げたベッドの前で、弱そうな番長さんがそれを売ってくれましたっけ。

……もう20年近く前の話です。まだ、伝統の番長さんはいるのでしょうか。どなたか最新情報を教えてください。

さて、十六進数を十進数に直すという作業は、人間とコンピュータとの間を結ぶ重要な仕事です。これがなければ、多くのコンピュータはタダの箱に逆もどりです。そうならないように、プログラマーはそのインターフェイスの役目を果たさなければなりません。

とりあえず、HLレジスタにある十六進数を十進数に変換し、一桁ずつメモリに入れるというプログラムを組んでみます。

```
MEMRY :   DB      0,0,0,0,0
          LD      HL,0FFFFH      ← HL＝変換したい十六進数
HEXDC :   LD      DE,MEMRY
          LD      BC,10000
          CALL    WARIS
          LD      BC,1000
          CALL    WARIS
          LD      BC,100
          CALL    WARIS
          LD      BC,10
          CALL    WARIS
          LD      A,L
          LD      (DE),A
          RET
WARIS :   XOR     A              ←「HL÷BC＝A･･･HL」を実行し、DE レジス
WARI1 :   SBC     HL,BC            タで示されるアドレスにAレジスタの値を
          JR      C,WARI2          入れる。DE レジスタは＋1される
          INC     A
          JR      WARI1
WARI2 :   ADD     HL,BC
          LD      (DE),A
          INC     DE
          RET
```

「HEXDC」はサブルーチンになっていますから、十進数にしたいデータをHL レジスタに入れてコールすれば、その都度「MEMRY」に十進データを得ることができます。このあとで「MEMRY」にあるデータを表示するのは、簡単なことでしょう。あるいは、このサブルーチンにある「WARIS」で、直接Aレジスタの値を表示するようにプログラムを直してもいいでしょう。

マドリッドといえば、マイヨル広場の近くに‘森の家’という安くて評判の食堂がありました。貧乏旅行者でも、ここに行けばバカ（肉）がバカバカ食べられると、その名声は遠くおフランスの町まで届いておりましたナ。

いつの日か、また放浪の旅をしたい……。

問 その42 BCD数値の加減算

ワシは海賊船の船長、グルメ・クックだ。

営業海域はカリブ海。ここには無数の宝島があり、欲に飢えた人間どもが金を持って乗り込んでくる。ウワッハッハ……。みな、ワシのお得意様よ。

近ごろでは、金が貯りすぎてとても数えきれん。そこで、コンピュータを使って残高を計算できるようにしようと思うのだ。もちろん、16ビットのレジスタなんかで間に合うわけがない。そういう場合、メモリを使うということも知っておる。

しかし、問題がないわけではない。それを画面にどうやって十進数にして表示するかがわからんのだ。0～$FFFF_H$（0～65535）までの表示法は最近覚えたのだが……。

イヤ、待てよ。できんということもないナ。メモリどうしの引き算（問その40参照）は知っているから、100 万で割り算をするなら、メモリに $0F4240_H$を入れて引き算を繰り返せばいいのだろう。

でも、面倒くさい話ではないか。いちいち割り算をするなんて。金がドンドン入ってくるのに、こんな面倒なことやってられるかってんだ。簡単に十進にする方法はないか。

カリブの海賊（ネズミーランド）

答

カリブの海賊？　まさか、浦安とかロスとかフロリダにいるという例の…。

確かに、あれだけの黄金財宝を数えるとなると、大変なことでしょう。いったい何桁になるのか見当もつきません。割り算のプログラムを組むにしても、割る数が１兆などという桁になると、それ自体を十六進数に変換するのが大変です。

そこで登場するのが、BCD（Binary Coded Decimal＝二進化十六進数）による計算方法です。これは、十六進数のうち 0～9 までを使用し、A～F は使用しないで計算しようという考え方です。

例えば、これまでは 12_Hといえば十進数では18でしたが、これを十進数の12と見なすのです。もちろん、見なすのはあなたですから、計算の相手に A～F を含んだ数値を使ってしまっては意味がありません。

この条件さえ守れば、プログラム的には簡単です。加減算命令のあとに「DAA 」と一言入れてやればいいのです。問（その40）の計算例と比べてみてください。

DATA0 ＋ DATA1

```
DATA0 :  DB    32H,20H,58H
DATA1 :  DB    13H,50H,88H
KEKKA :  DB    0,0,0
ADD01 :  LD    DE,DATA0+2
         LD    HL,DATA1+2
         LD    IX,KEKKA+2
         OR    A
         LD    B,3
ADDLP :  LD    A,(DE)
         ADC   A,(HL)
         DAA
         LD    (IX+0),A
         DEC   DE
         DEC   HL
         DEC   IX
         DJNZ  ADDLP
         RET
```

DATA0 － DATA1

```
DATA0 :  DB    32H,20H,58H
DATA1 :  DB    13H,50H,88H
KEKKA :  DB    0,0,0
SUB01 :  LD    DE,DATA0+2
         LD    HL,DATA1+2
         LD    IX,KEKKA+2
         OR    A
         LD    B,3            ←B=桁数
SUBLP :  LD    A,(DE)
         SBC   A,(HL)
         DAA
         LD    (IX+0),A
         DEC   DE
         DEC   HL
         DEC   IX
         DJNZ  SUBLP
         RET
```

計算の終了した時点で、「KEKKA」にはそのまま十進数として読めるような形で答が入っています。画面に表示する時には、それぞれの値を上位／下位に分けて表示すればいいのです。

上位の値をAレジスタへ

```
LD      A,(HL)
RRCA
RRCA
RRCA
RRCA
AND     00001111B
```

下位の値をAレジスタへ

```
LD      A,(HL)
AND     00001111B
```

注）どちらもHLレジスタで表示したい数値のあるアドレスを指定したものとする。

これで、割り算による手間が省けるとともに、桁（兆の位とか億の位…等）を意識しないで数値を表示することができるわけです。

ところで、カリブの海賊は有名だけど、ネズミーランドっていうのは？？？

問 その43 データの左右反転

初めての質問です。雑誌に投稿するのが趣味の高校３年ですが、これからは受験勉強のため投稿はしばらく休まなければなりません。でも、これまでに某誌に２回も自分の作品が発表されたことがあるんですよ（BASIC＋ マシン語だよ〜ン)。

マシン語は、おもにキャラクタの表示部分なんだけど、左右反転したパターンを見るたびに、メモリがもったいないなァと思います。

インストラクション表で、左右反転ができるような命令を捜してみましたが、どうもなさそうです。

なんとか、この無駄を解決する方法はありませんか。できれば、受験の前にこのモヤモヤを取り除きたいなァ。

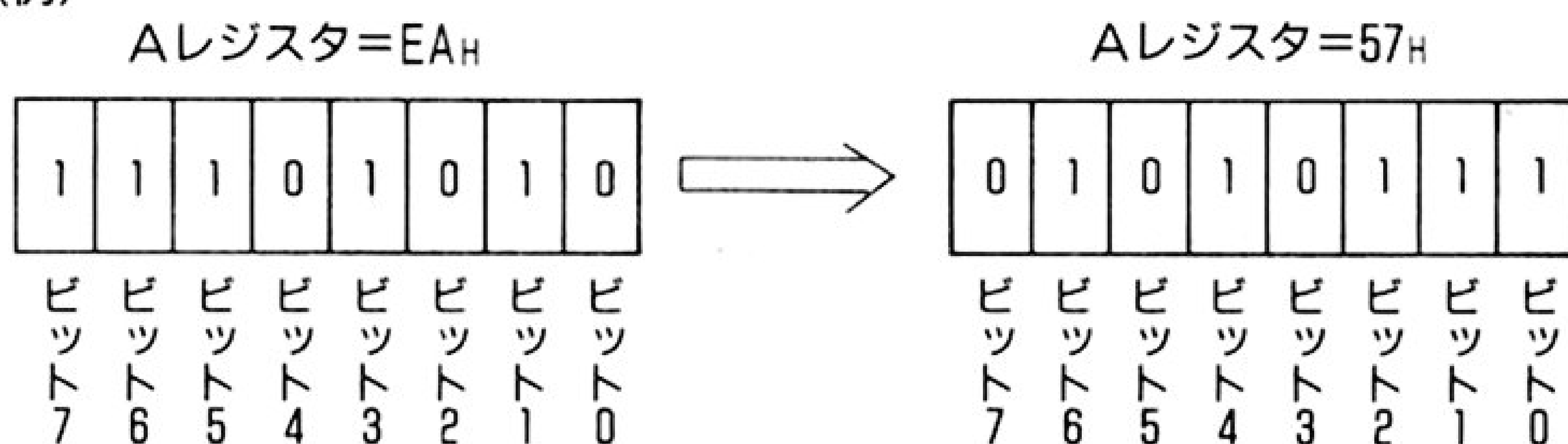
投稿マニア（徳島）

答

さすが、投稿マニアを自称するだけあって、鋭い質問ですね。データは少なく、プログラムは短く、そして速く……マシン語の技術の差は、ここに現れるのです。

すでに、このことに目覚めているからこそ、これだけの質問がでるのでしょう。おそらく、受験勉強がなければ自力で解決できるだけの実力があるはずです。

とりあえず、質問の内容が具体的でないので、例をあげて解決することにします。



左右反転のパターンを作るには、１つのデータを図のように左右対象に入れ換えなければなりません。残念ながら、この命令はいくらインストラクション表を捜してもありません。また、論理演算命令を繰り返しても作ることはでき

ません。

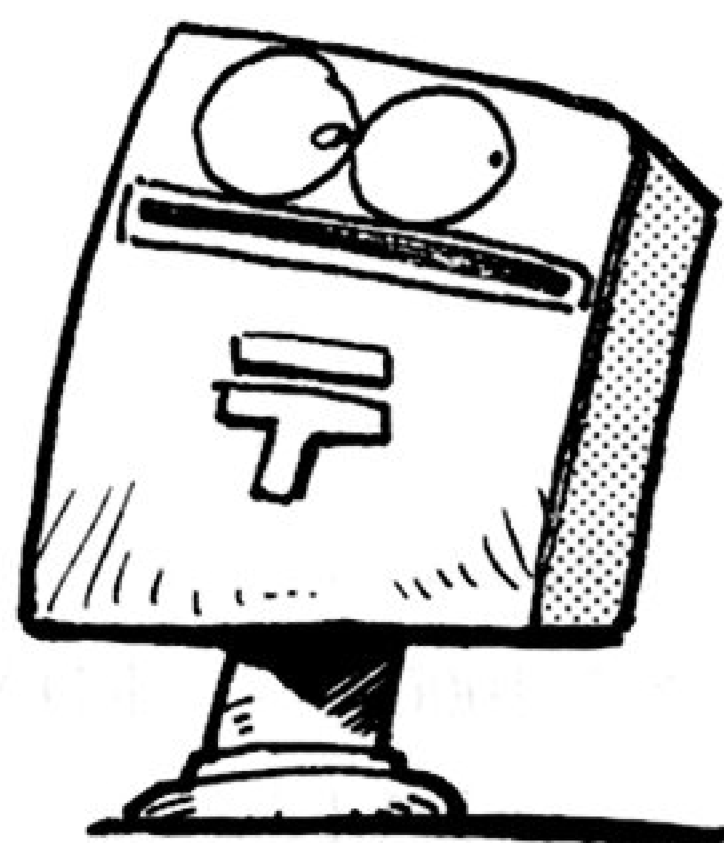
ということは、プログラムで反転データを作るしかないわけです。そこで、キャリーフラグを利用して次のようにします。

```
        LD      B,8
LOOP :  RLCA
        RR      C
        DJNZ    LOOP
        LD      A,C
```

Aレジスタの値を左回りにローテートさせると、7ビット目の値がキャリーフラグに入ります。次に、キャリーフラグの値がCレジスタの7ビット目に入るようにCレジスタを右にローテートさせます。これを8回繰り返すわけです。

キャリーフラグの動きは、インストラクション表を見ればわかるでしょう。問題は少々時間がかかることで、速度を追求する場合には利用できないかもしれません。それを考慮した上で使えば、結構役に立てられるはずです。

でも、この程度の内容の命令は最初から用意しておいて欲しいものですネ。



問 その44

実用できる左右反転プログラム

P。これは「ピー」と読みまP。こちらは、ミニ独立国家ピータンでP。わが国の憲法はただひとつ、言葉の最後にPを付けることでP。

では、質問でP。最近、データを反転するテクニック(問その43参照) を覚えたのですが、どうも実用性に乏しいような気がするのでP。その理由は速度でP。1バイトのデータを反転するのに、

```
        LD      B,8       ←7
LOOP :  RLCA              ←4×8=32
        RR      C         ←8×8=64
        DJNZ    LOOP      ←13×7+8=99
        LD      A,C       ←4
```

で、トータル 206ステートもかかっていまP。たった1バイトならともかく、大量のデータを反転するとなると問題でP。しかも、そのためにBCレジスタを破壊しなければならないのでP。

これでは、現実には使いたくても使えないということになってしまいまP。もっとうまい方法があると、とても便利でハッピーな気分になれそうでP。

ピータン国王(岩手)

答

さすが、国王。単にプログラム・テクニックを知っただけでは満足せず、そこに実用性を求めていまP。おそれ入りましたでP。

確かに、1バイトを反転するのに 206ステートもかかるのは問題でしょう。グラフィックなどはデータの山ですから、リアルタイム・ゲームに応用するのはほとんど不可能とも言えます。

そこで、少々メモリは必要ですが、とっておきの秘法を紹介します。

このプログラムで注意すべき点は、「RDATA」のスタートアドレスを100_Hバイト単位で設定する(アドレス下位=00_H)ということです。ここでは、あえてORG命令で 1000_Hから「RDATA」が開始するようにしましたが、このルールさえ守れば「RDATA」はどこに置いても構いません。

実行速度は1バイトにつき26ステートですから、180 ステートも速くなったことになります。また、Aレジスタを反転するのに他のレジスタを必要としま

```
            ORG     1000H

RDATA:      DB      00H,80H,40H,0C0H,20H,0A0H,60H,0E0H
            DB      10H,90H,50H,0D0H,30H,0B0H,70H,0F0H
            DB      08H,88H,48H,0C8H,28H,0A8H,68H,0E8H
            DB      18H,98H,58H,0D8H,38H,0B8H,78H,0F8H
            DB      04H,84H,44H,0C4H,24H,0A4H,64H,0E4H
            DB      14H,94H,54H,0D4H,34H,0B4H,74H,0F4H
            DB      0CH,8CH,4CH,0CCH,2CH,0ACH,6CH,0ECH
            DB      1CH,9CH,5CH,0DCH,3CH,0BCH,7CH,0FCH
            DB      02H,82H,42H,0C2H,22H,0A2H,62H,0E2H
            DB      12H,92H,52H,0D2H,32H,0B2H,72H,0F2H
            DB      0AH,8AH,4AH,0CAH,2AH,0AAH,6AH,0EAH
            DB      1AH,9AH,5AH,0DAH,3AH,0BAH,7AH,0FAH
            DB      06H,86H,46H,0C6H,26H,0A6H,66H,0E6H
            DB      16H,96H,56H,0D6H,36H,0B6H,76H,0F6H
            DB      0EH,8EH,4EH,0CEH,2EH,0AEH,6EH,0EEH
            DB      1EH,9EH,5EH,0DEH,3EH,0BEH,7EH,0FEH
            DB      01H,81H,41H,0C1H,21H,0A1H,61H,0E1H
            DB      11H,91H,51H,0D1H,31H,0B1H,71H,0F1H
            DB      09H,89H,49H,0C9H,29H,0A9H,69H,0E9H
            DB      19H,99H,59H,0D9H,39H,0B9H,79H,0F9H
            DB      05H,85H,45H,0C5H,25H,0A5H,65H,0E5H
            DB      15H,95H,55H,0D5H,35H,0B5H,75H,0F5H
            DB      0DH,8DH,4DH,0CDH,2DH,0ADH,6DH,0EDH
            DB      1DH,9DH,5DH,0DDH,3DH,0BDH,7DH,0FDH
            DB      03H,83H,43H,0C3H,23H,0A3H,63H,0E3H
            DB      13H,93H,53H,0D3H,33H,0B3H,73H,0F3H
            DB      0BH,8BH,4BH,0CBH,2BH,0ABH,6BH,0EBH
            DB      1BH,9BH,5BH,0DBH,3BH,0BBH,7BH,0FBH
            DB      07H,87H,47H,0C7H,27H,0A7H,67H,0E7H
            DB      17H,97H,57H,0D7H,37H,0B7H,77H,0F7H
            DB      0FH,8FH,4FH,0CFH,2FH,0AFH,6FH,0EFH
            DB      1FH,9FH,5FH,0DFH,3FH,0BFH,7FH,0FFH
                    :

            LD      (GETRV+1),A
GETRV:      LD      A,(RDATA)
                    :
```

せんので、ループの中にこのまま組み入れることも可能です。

アイデアがシンプルなだけ、実用性は逆に高くなったと言えるでしょう。大いに活用してください。

では、国王。バイＰ……。

問その45 Hフラグ、Nフラグとは

グワッハッハ……!!

オレ様は地獄の番人、閻魔大王だ。グワッハッハ……!!

オレ様の前でウソなどついてみろ。アッという間に、この大きな'やっとこ'で舌を抜いてしまうからナ。オレ様は、人間の話す言葉から、それがウソか真実か、簡単に見抜くことができるのだ。

この間も、「手みやげに金の延べ板を持参してきました」と、オレ様に鉛に金メッキをしたまがい物を渡そうとしたヤツがおってな。もちろん、血の池経由の針山行きよ。バカ者めが……。

ところが、このオレ様にも最近弱点ができてしまったのだ。それは、なんとマシン語を話題にする相手のことだ。オレ様も必死になってマシン語を勉強したのだが、なかなか思うように上達せんのだよ。

今日も、「フラグの中にはHフラグやNフラグなど、ユーザーには関係のないフラグがある」などと言われてしまい、どうにもわからんで困っておる。なんとか、オレ様の役目も理解して、親切にご指導願いたい。

閻魔大王クッパ（地獄一丁目）

答

こわそーッ。でも、ここでウソなど教えたら、いつの日かお目にかからなければならなくなった時、血の池経由の針山行きなどとされてしまいそうです。それだけは、なんとしても避けなければ……。

では、まずHフラグとNフラグの内容について説明をします。

・H（ハーフキャリー）フラグ

加減算などニモニック表のHフラグが「↑↓」となっている命令を実行後、下位4ビットから見て、上位4ビットからの桁借り、あるいは上位4ビットへの桁上がりがあった場合にセットされます。上位／下位単独の変化の場合はリセットされます。

・N（加／減算）フラグ

減算命令を実行するとセットされ、加算命令を実行するとリセットされます。

両方のフラグとも、厳密にはここに示した命令以外の命令によっても変化しますが、それらは無意味な変化です。というのは、これらはユーザーが利用するためにあるのではなく、BCD演算におけるDAA命令のために用意されているからです。

したがって、BCD演算に関係のない命令では、どのようなフラグ変化をしようと基本的に価値がありません。ユーザーが利用できないようになっているのも、特に必要がないからだといえるでしょう。

しかし、DAA命令によっても変化するHフラグは、利用できるなら利用したい時がないわけでもありません。例えば、特定の桁が変化したかどうかを調べるという場合など、Hフラグがキャリーフラグとともに利用できると便利です。

このような場合でも、Hフラグがフラグレジスタの中にある限り、どうやっても調べることはできません。あまり感心できる方法ではありませんが、次のようにすれば一応Hフラグを調べられます。

```
PUSH    AF
POP     BC          ←フラグレジスタの値をCレジスタへ
BIT     4,C         ←Hフラグのチェック
JR      NZ,xxxx
         :
```

この方法であれば、もちろんNフラグも調べることはできますが、いずれにしてもそれほど必要になることはないでしょう。要するに、やむを得ない時には、こういった奥の手もあると覚えておけばいいという程度のことです。

これらは、決してウソの説明ではありません。本当に、本当ですとも……閻魔大王クッパさま……。

問 その46

いち度のCP命令で5つの条件判断

アー、忙しい忙しい。とにかく、この日は忙しい……。

わしか、わしはサンタクロースじゃよ。毎年、12月24日の夜には世界中の子供たちにプレゼントを配らなければならないのじゃ。そして、いつものことじゃが悩むのは配る順序についてじゃ。下手をすると夜が明けてしまうからな。

そこで、最近はコンピュータを導入して合理化を図っておるのじゃ。「エーッ!!」と驚く人もいるかもしれないが、このプログラム次第でわしの真価が問われるのじゃから、気合いも入ろうというもの。

いま、ここに5種類のプレゼントが用意されているとしよう。子供たちは、サンタ・プレゼント方程式により、皆ある数値を持っておる。そして、その値によりどのプレゼントがもらえるかが決まるようになっておるのだ。

その値とは、0、1、2～$7F_H$、80_H、81_H～FF_H の5種類で、ここからそれぞれのプレゼントルーチンへとジャンプさせるわけじゃ。テーブルを作ってジャンプさせるには不便だし、やっぱりCP命令で1つひとつ分岐させるしか手はないかのう……。

とにかく、この条件分岐は何度も使用するので、できるだけ速く分岐させたいというのが、わしの希望じゃ。

サンタ日本代理人・黒須三太（富士山）

答

サンタの世界にもコンピュータが入り込んでいるとは、正直のところ驚かずにはいられません。そして、ここでも要求されているのはスピードです。まさに、スピードを制す者は世界を制すという感じです。

ちなみに、質問の内容をプログラムにすると次のようになっているのでしょうか。

```
            CP    1
            JP    C,SANT0
            JP    Z,SANT1
            CP    80H
            JP    C,SANT2
            JP    Z,SANT3
SANT 4 :          :
                  :
```

これでも悪いということはありません。どちらかというと、非常に一般的といえるでしょう。しかし、ここはひとつ 0、1、2〜$7F_H$、80_H、81_H〜FF_H という、条件分岐の基になっている値に注目してみたいのです。

この値が偶然に付けられたのか、それとも特別な意図があって付けられたのかは不明ですが、一度のCP命令で5つの判定ができる実に有効な数値なのです。では、それぞれの値について、「CP　1」によるフラグの変化を示してみます。なお、H（ハーフキャリー）フラグについては、条件分岐に利用できないので省略してあります。

	サイン	ゼロ	P／V	キャリー
0	1(M)	0(NZ)	0(PO)	1(C)
1	0(P)	1(Z)	0(PO)	0(NC)
2〜$7F_H$	0(P)	0(NZ)	0(PO)	0(NC)
80_H	0(P)	0(NZ)	1(PE)	0(NC)
81_H〜FF_H	1(M)	0(NZ)	0(PO)	0(NC)

どうですか。4つのフラグをうまく利用すれば、一回のCP命令ですべての分岐先へジャンプできそうですね。ただし、分岐の順序を間違えると、せっかく変化してくれたフラグが役に立ちません。次の例を参考に、注意してジャンプさせてください。

```
        CP      1
        JP      C,SANT0
        JP      Z,SANT1
        JP      PE,SANT3
        JP      P,SANT2
SANT4 :         :
                :
```

たった2バイト、7ステートの節約にしかなりませんが、要はその心意気が大切なのです。10万回実行すれば、70万ステートの節約になるのですから。さらに、この分岐に優先順位（分岐する度合が多い方を先に分岐させる）をつけるところまで気を配れるようになると、スピード重視も本物です。もちろん、その場合でも先のフラグ変化表を確認するのは当然ですが……。

せっかくあるフラグですから、数値を特に連続させなくてもいい場合は、このサンタ的分岐法を大いに活用しましょう。

問 その47 「CP HL, DE」を実現

拝啓。春光うららかな今日このごろ、皆様にはますますご隆盛大慶に存じます。

わが輩は、明治生まれの真空管技術者でござりまする。わが輩も、一時期は時代の最先端をいく花形技術者としてもてはやされたのでござりまするが、それも今や過去の夢物語であります。

そこで一念発起、パソコンを購入しマシン語なるものにチャレンジしたのでござりまする。結構、色々な命令があるものでござりますな。これまでの調べで、8ビットを操作する命令に比べて、16ビットを操作する命令が少ないことも判明しております。

8ビットマシンである以上、ある程度はやむを得ないのかもしれぬが、それにしても16ビット用のCP命令がないのはどういうことでござろうか……。

つまり、「CP　HL,DE」とか「CP　HL,BC」という命令が欲しいのでござりまする。いくら8ビットマシンとはいえ、この程度の能力がないようでは、なにかと不便で仕方がないのでござりまするな。よろしくお頼もうすでござる。　敬具

明治キメラ（長野）

答

これはこれは、ご丁寧なご質問ありがとうでござりまする。パソコンという共通の話題がある限り、「明治は遠くなりにけり」なんて死語でござりまするな。

確かに、8ビット用のCP命令があるのに、16ビット用のCP命令がないというのは不便なことかもしれません。例えば、現在の所持金が10000ゴールドで、1000ゴールドの剣が買えるかどうかを調べる、なんていう場合もありますから。

しかし、命令として存在しないものは自力で作ればいいのです。そもそも大した命令がないのがマシン語ですから、ないものはプログラムで作るのが当然ともいえます。さっそく、質問にある2つの命令をサブルーチンとして作ってみました。

```
CPHDE :   PUSH   HL
          OR     A
          SBC    HL,DE
          POP    HL
          RET
```

```
CPHBC :   PUSH   HL
          OR     A
          SBC    HL,BC
          POP    HL
          RET
```

プログラム的にはどちらも同じようなものです。変化するのはフラグだけですから、いつでも16ビットのCP命令として利用することができます。8ビットのCP命令と違うところは、H(ハーフキャリー)フラグが不定であるという点ですが、このフラグは基本的にユーザーが利用するものではないので、実用上は8ビットのCP命令と同じと考えていいでしょう。

もし、裏レジスタのフラグ(F')を破壊しても構わないというのであれば、次のようにすることで実行速度を少しアップすることができます。

```
CPHDE :  OR     A
         SBC    HL,DE
         EX     AF,AF'
         ADD    HL,DE
         EX     AF,AF'
         RET
```

```
CPHBC :  OR     A
         SBC    HL,BC
         EX     AF,AF'
         ADD    HL,BC
         EX     AF,AF'
         RET
```

PUSH／POP命令(21ステート)を利用したものに比べて、メモリは1バイト多く使用しますが、トータルで2ステート速くなっています。このこと自体は特筆すべきことではなく、速くするという意識の表現という程度です。プログラムに支障がなければ、こちらのほうを利用しましょう。現実的には、裏レジスタを使用してもフラグまで保存しておくということは余りないので、特殊なケース以外はほとんど問題ないはずです。

では、がんばってくださいでござりまする……。明治キメラ殿。



問その48 「CP BC, DE」を実現

京都〜ォ、大原三千院……。

うちは京の舞妓どす。優雅に歌など歌っている姿を想像して、うちの美しさを味わっておくれやす。

でも、これはマイコンが趣味でドス（DOS＝Disk Operating System）を作りたいという気持ちが高まって、いつの間にか自分のことを舞妓はんと思うようになったのが本当の話どす。

勘違いされる前に宣言しておきやすけど、一応正真正銘の女の子どすよってに、気色ワル〜イとかは言わはらんようにお願いしやす。ちなみに、うちの話す舞妓はん言葉もメチャクチャどすェ。なにしろ、生まれは関東どすから……。

いま、あるプログラムで「CP　BC,DE」という内容を実行したいのどすが、うちの実力ではどうにもわからんのどす。HL レジスタとの比較ならば、最近覚えたばかりのプログラム（問その47参照）があるのどすが、HL レジスタは使用中どす。

こういう場合、いったいどうしたらいいのどす？　困ってしまうどす。ア〜ァ、それにしてもマイコン以上に舞妓はん言葉は難しいどすねェ。

ニセ舞妓（京都）

答

言葉の最後に「どす」を付ければ舞妓さんというのも、なにかモノスゴイ勘違いのような気がしますどす。とりあえずは、正真正銘の女の子ということが唯一の救いかもしれませんどす。

さて、この命令に相当するプログラムは、次のようにすれば問（その47）とほぼ同じ感覚で作ることができます。

```
CPBDE :  PUSH   HL
         LD     H,B
         LD     L,C
         OR     A
         SBC    HL,DE
         POP    HL
         RET
```

しかし、問（その47）にも当てはまることですが、こういったケースではゼロフラグは特に必要としないことが多いものです。つまり、キャリーフラグに

よる大小の判断、数学的にいえばイコールを含んでより大きい（≧）か小さい（<）かの判断だけでいいというケースです。

このような場合、Aレジスタの値は破壊されますが、非常に簡単なプログラムで大小を判断することができます。

```
CPBDE :  LD     A,C
         SUB    E
         LD     A,B
         SBC    A,D
         RET
```

ここではサブルーチンとして記述しましたが、実質4バイト（12ステート）ですからサブルーチン化しないほうがいいのは当然です。また、この方法ではレジスタとの比較だけでなく、16ビットの数値と直接比較することもできます。例えば、BCレジスタと 1234_H を比較しようという場合、次のようにすればいいのです。

```
LD      A,C
SUB     34H
LD      A,B
SBC     A,12H
```

便利さのあまり、ゼロフラグが正確にセットされないということを忘れてしまうと大変です。その点だけは、常に注意するようにしてくださいどす。

問その49 ペアレジスタのNEG命令

ここはエジンバラ……。イギリス北部、スコットランドにある美しい町です。申し遅れましたが、ボクはイギリス国内をヒッチハイクによって旅している者です。

慣れてしまえばどうということのないヒッチハイクも、最初は親指を立てて右手を上げるのにとても勇気が入りました。一台、また一台……と車が通過するたびに、ボクには絶対に止まってくれないのではないかという不安感が頭をかすめたものです。

1時間ほどして、ようやく一台の乗用車が止まってくれた時は、ボーッとして一瞬われを忘れたほどです。でも、それがきっかけとなってヒッチハイクのコツがわかり、今ではマナーをわきまえた一流のヒッチハイカーです。

それはいいとして、忘れられないのが日本に置いてきたコンピュータのこと。というより、せっかく覚えたマシン語のことです。だから、頭の中はいつもマシン語でいっぱいです。車が止まってくれない時は、もちろんマシン語のことを考えています。

「NEG」というマイナスの値を作る命令がありますね。ある時、あれの16ビット版はどうやったらいいか。これを考えていたら、右手を上げるのをすっかり忘れてしまったことがあります。

おかげで、車は止まらないわ、雨は降ってくるわ……で、さんざんな目にあってしまいました。

どうか、気分よくヒッチハイクができるように教えてください。

ヒッチくん（エジンバラ）

答

日本ではヒッチハイクの習慣はあまりありませんが、ヨーロッパなどでは若者の移動手段として立派に市民権を得ています。目的地にいつ到着するかわからないという不確実な点も、若者の気ままな旅に合っているのかもしれません。

ヒッチハイクのコツは、まず郊外の街道筋まで出るということ。例えば、銀座や新宿のような場所では成功しません。そして、車が止まってくれたら、急いで車のほうへ走り寄り、行き先をハッキリと言います。行き先が違えば、残念ながら乗るのはあきらめなければなりません。運よく乗せてもらうことができたなら、相手に不快感や不安感を与えないようにするのも大切なことです。乗せて楽しかったと思われるようになれば、もう一流のヒッチハイカーでしょう。

私は五円玉をたくさん用意しておいて、降りる時に記念に一枚渡しました。外国には穴の開いたコインが少ないので、結構喜ばれたものです。

おっと、あまりヒッチハイクの秘伝ばかり書いていると、肝心の質問を忘れてしまいそうなので、ここではこれくらいにしておきましょう。

8ビットの「NEG」は問(その11)で紹介しましたが、16ビット用の「NEG」は命令として存在していません。そこで、プログラムで作ることになります。

```
NEG16 :  DEC    HL
CPL16 :  LD     A,L
         CPL
         LD     L,A
         LD     A,H
         CPL
         LD     H,A
         RET
```

ここではHLレジスタについて記述しましたが、BC／DEレジスタについても同様のことが可能です。また、プログラムのラベルを見てお分かりのように、最初の「DEC　HL」を省くことにより「 CPL」の16ビット版とすることができます。

ちなみに、ビット単位で「 CPL」と「 NEG」の違いを見ると、次のようになります。Aレジスタの値は 00001111B＝$0F_H$ と仮定します。

CPL ： 11110000B＝$F0_H$　(数値上は「XOR　FFH」と同じだが、フラグ変化が違う)

NEG ： 11110001B＝$F1_H$　(CPLした値に＋1したもの)

両者とも、内容をよく理解した上で使用することが大切です。フラグ変化の違いなどもよく確認しておきましょう。気分よくヒッチハイクするためにも……。

問 その50

ジャンプ代わりにリターン

先手５八玉……。

どうだ!!　これがワシの発明した戦法「玉先導」の一手目だ。その先はまだ秘密で教えられないが、この手で動揺した棋士は皆あわてふためいて、結局は負けてしまうことになるのだ。ワッハッハッハ!!

ワシはプロの棋士、相手もプロ……。といっても、相手のプロはプログラムで動くコンピュータという意味だがな。もちろん、ワシがプロの棋士というのも、プログラムで動く棋士という意味だ。

おっと、正確にはワシはそのプログラマーというわけだ。できるだけ強いプログラムにするには、メモリを効率よく使って思考ルーチンを強化しなければならぬ。実は、プログラムの一部に50個所にジャンプする部分がある。当然、テーブルは使っている。

```
TABLE :  PRO00,PRO01,……,PRO48,PRO49

MAINP :          ⋮
         LD      HL,TABLE
         RLCA
         LD      E,A
         LD      D,0
         ADD     HL,DE
         LD      E,(HL)
         INC     HL
         LD      D,(HL)
         EX      DE,HL
         JP      (HL)
```

プログラムの概略は以上なのだが、それぞれのルーチンからは「MAINP」へジャンプしてもどってくるようになっておるのだ。単純に計算しても「JP　MAINP」が50個はあるということだ。そのほかにも、条件分岐でもどるものもあるので、実際には100個所以上から「MAINP」へジャンプしてくるのだ。

こういったルーチンがいくつかあるので、これに要するメモリもバカにならない。なんとか省メモリ化できないだろうか。

白夜の棋士（静岡）

答

コンピュータで本当に強い将棋プログラムができたら、つまり本物のプロ棋

士と互角に戦えるようなレベルになったら、こんな痛快なことはないでしょうね。

なぜかというと、そこまで強いプログラムになると、理論上は常に最善手を指してくるからです。そうすると、敗着の手を指す恐れのある人間の頭では、アッという間にコンピュータに勝てなくなってしまうでしょう。では、そんなコンピュータどうしの勝負はどうなるのか、それを早く見てみたいものです。

しかし、現在のコンピュータのレベルでは、大局感という要素があまりないので、当分はお遊びソフトの域を出るのは難しそうです。

さて、質問にあるようなメインプログラムへの大量ジャンプですが、こういったケースにはアチコチで遭遇します。省メモリを目指す場合には、次のようなテクニックを覚えておくと便利です。

```
TABLE :  PRO00,PRO01,……,PRO48,PRO49

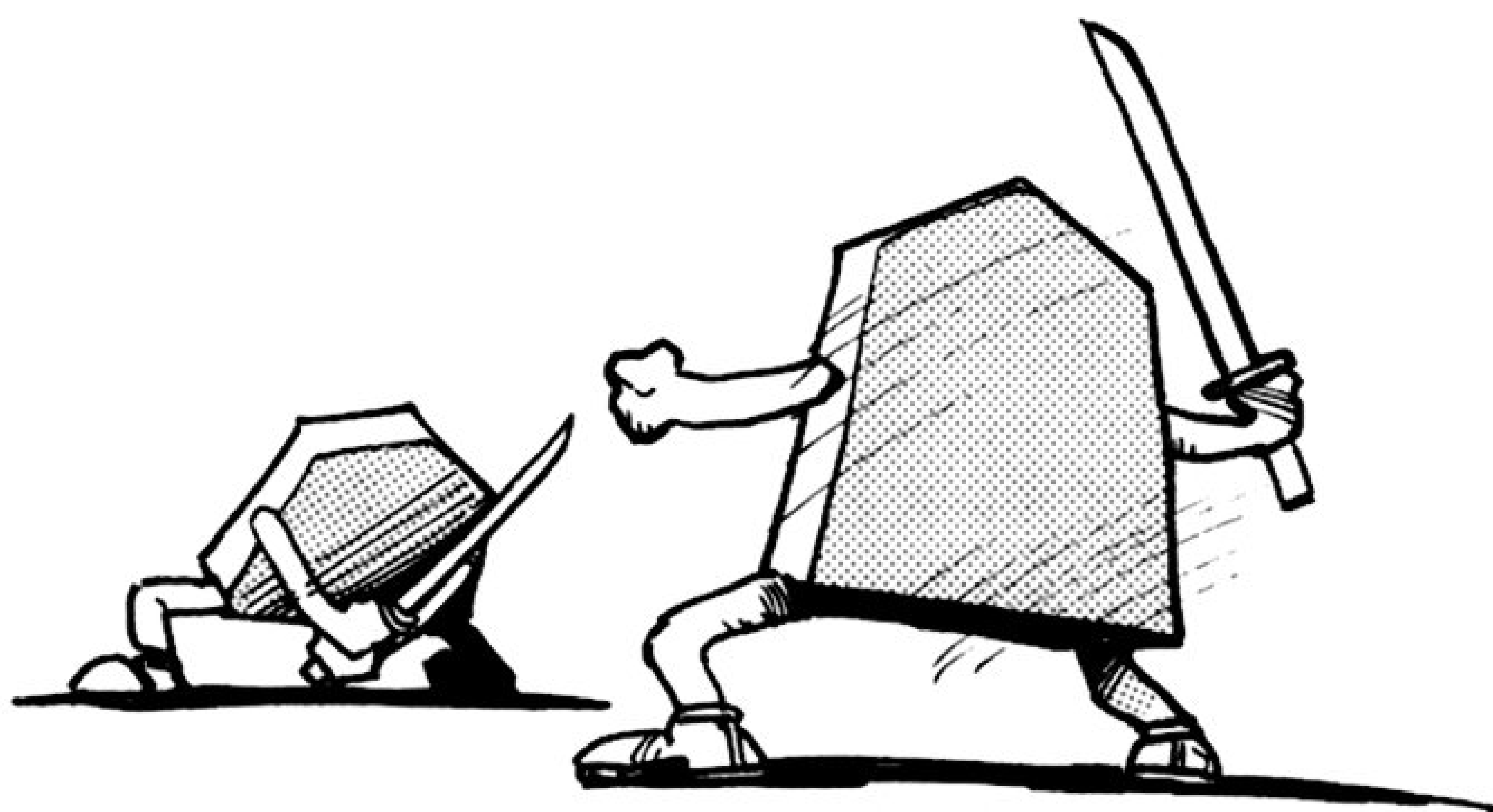
MAINP :          ⋮
         LD      HL,MAINP      ← MAINPのアドレスを、
         PUSH    HL              もどりアドレスとして
         LD      HL,TABLE        スタックへ入れておく
         RLCA
         LD      E,A
         LD      D,0
         ADD     HL,DE
         LD      E,(HL)
         INC     HL
         LD      D,(HL)
         EX      DE,HL
         JP      (HL)
```

つまり、「MAINP」をリターンアドレスとしてスタックへ退避しておくのです。こうすれば、それぞれのルーチンからJP命令でなくRET命令で、「MAINP」へジャンプさせることができます。

この時注意しなければならないのは、各ルーチン先で、場合によっては「MAINP」へもどらないことがある場合です。このような場合、スタックへリターンアドレスを退避してあることを忘れてしまうと、SPが狂って暴走する

危険性があります。したがって、RET命令で直接「MAINP」へもどらない時は、SP合わせのダミーとして「POP AF」などを実行するか、「INC　SP」「INC　SP」としてSPを元にもどす必要があります。

このことにさえ注意すれば、このテクニックの活用の場は結構多いはずです。うまく活用して、本当に名人級の将棋ソフトを作ってください……。



問 その51 「LD HL, SP」を実現

ホワァ〜ァ……。眠たい眠たい……。

プログラマーは眠たい。とにかく眠たい。おっと、これは失礼。私は、某ソフトハウスでゲーム制作のプログラム・アシスタントをしている者です。

アシスタントといっても、自分の受け持ったルーチンは責任を持って組まなければなりませんから、手伝いというような甘い考えは通用しません。しかし、恥ずかしながら私はプログラマーとしてはまだまだ未熟者です。

ニモニックの中に「LD　SP,HL」「LD　SP,IX」「LD　SP,IY」という命令がありますが、この逆の命令はありません。

SP（スタックポインタ）に値を設定するだけなら、直接「LD　SP,xxxx」という命令があるので、このような命令は使い道がないような気がしますが。いったい、SPをどうしたい場合にこれを使うのですか。

それにしても、ホワァ〜ァ……眠たい眠たい……。

眠りプログラマー（福島）

答

ホワァ〜ァ……眠たいのは、こちらも同じこと。まったく、いくら寝ても眠たさは解消しません。今も、必死に眠たさをこらえて質問に答えているところです……。

まず、「LD　SP,HL」等の反対の命令、すなわち「LD　HL,SP」ですが、これはSPを退避させるような感覚で簡単に作ることができます。

```
        LD    (LHLSP+1),SP     ← 4バイト（20ステート）
LHLSP:  LD    HL,0
```

問（その37）や問（その39）でスタックエリアの特殊な利用法がありましたが、その時に同様な方法でSPを退避させていました。しかし、SPというのはスタックポインタであると同時に、16ビットのレジスタでもあります。もう少しエレガントに扱ってあげましょう。

```
LD      HL,0
ADD     HL,SP
```

←1バイト（11ステート）

実行に要するステート数、そして必要なメモリ数も最初の方法より少なく、プログラムとしてキレイに仕上がっています。この例のように、レジスタ SP は「PUSH／POP」ができないことを除けば、単なる16ビットレジスタとしての利用も可能です。

ただし、SP に別の数値を代入して用いる場合は、最初の方法で SP を退避し、DI（割込み禁止）をかけてから利用しなければなりません。このあたりの注意は問（その37）にあることと同じです。

次に、「LD　SP,HL」などをどのようなケースに使うかですが、通常はあまり使うことはありません。質問にあるように、直接 SP に値を設定すればいいからです。

しかし、先ほどの例のように SP をレジスタとして用いている場合、16ビットを一度に移動できるので役に立つこともあります。例えば、「HL＝HL＋IX」という内容を実行したいと仮定しましょう。当然、このような命令はありませんから、PUSH/POP 命令で IX を DE に移すなどしてから加算しなければなりません。そこで、次のようにすることもできます。

```
PUSH   IX      15ステート
POP    DE      10ステート
ADD    HL,DE
```

⇨

```
LD     SP,IX   10ステート
ADD    HL,SP
```

このようなケースはめったにあるわけではないし、割込みの禁止やスタックポインタとしての SP の値を退避しておくことも前提です。通常のプログラムにおいては SP をスタックポインタとして使っているわけですから、これらの命令が活躍する場はあまりナイと考えたほうがいいでしょう。

ホワァ〜ァ……。本当に眠たくなってしまった、ムニャ……。

問 その52

CPIR命令について

ゲ、ゲ、ゲゲゲのゲーッ……。

近ごろ、ウイルスというコンピュータを悩ます妖怪がはびこっているそうだが、そんなことで困っている人は、この正義のコンピュータ妖怪「ゲゲゲのQ太郎」に連絡してください。

連絡方法は、お近くのゲゲゲ・ネットワークへ申し込めばいいようになっています。あなたのシステムに入り込んで、即座にウイルス妖怪をやっつけてあげましょう。

……という看板を出したのはいいけれど、実はまだ本物のウイルス妖怪にはお目にかかったことがありません。つまり、この看板は「出たとこ勝負」のバクチなのです。とはいえ、とりあえずは正体不明の妖怪対策も考えておかなければなりません。

たぶん、ウイルス妖怪退治には値をサーチする命令が欠かせないでしょう。Aレジスタの値をサーチする命令として、次のような命令がありますね。

CPI　：Aと（HL）を比較し、HL←HL＋1、BC←BC−1 をする
CPIR　：A＝（HL）となるか、BC=0となるまでCPI命令を繰り返す
CPD　：Aと（HL）を比較し、HL←HL−1、BC←BC−1 をする
CPDR　：A＝（HL）となるか、BC=0となるまでCPD命令を繰り返す

これらの命令について、どうも実行結果やフラグの見方がよくわかりません。コンピュータの正義のために、正しい利用法を教えてください。

ゲゲゲのQ太郎（正マ界）

答

ゲゲゲのQ太郎……？　どこかで聞いたような名前です。頼りになる妖怪と頼りにならないお化けが合体したような、なんとも妙な雰囲気が漂っていますが、質問からするとまだまだ実力不明の妖怪のようです。

基本的にCPI (CPIR) 命令とCPD (CPDR) 命令とは、HLレジスタの動きが逆になるだけですから、ここではCPI (CPIR) 命令について説明をしていきます。

では、まずCPI命令を実行するとフラグ以外のレジスタがどのように変化するのかを確認しましょう。

```
HL ← HL+1
BC ← BC-1
```

これは、Aレジスタと(HL)が同じであろうとなかろうと、その結果に関係なく実行されます。このことは、当然CPIR命令についても同じです。

したがって、CPIR命令でA=(HL)となった場合、その時(命令実行後)のHLレジスタの値は+1されていることになります。もし、A=(HL)となっているHLレジスタを求めたいのであれば、CPIR命令の後で「DEC　HL」を実行しなければなりません。

次に、フラグ変化についてですが、これらの命令は一見すると自動的な連続CP命令に見えますが、フラグについては同一ではありません。まず、キャリーフラグがまったく変化しません。したがって、CPI命令後に大小の判断をキャリーフラグで行うことはできないわけです。

また、A=(HL)になった場合にはゼロフラグが立ちます。例えば、CPIR命令を実行した後で、ゼロフラグが立っていればA=(HL)となって命令を終了したことになり、ゼロフラグが立っていなければBC=0となって命令を終了したと判断できます。

では、A=(HL)となっている回数をBCレジスタの範囲だけ数える場合はどうしたらいいでしょうか。かりにDEレジスタをその回数カウンタとすると、前述の判断方法によるプログラムでは次のようになります。

```
         LD      DE,0
CLOOP :  CPIR
         JR      NZ,CPEND
         INC     DE
         JR      CLOOP
CPEND :           :
```

……が、このプログラムには大きなミスがあることに気がついたでしょうか。よく見ると、A=(HL)とBC=0が同時に起こった時を見逃しています。ゼロフラグはBC=0となっても最後にA=(HL)となっていれば立ちますから、こ

のままでは再び10000_H 回のCPIR命令を実行してしまうことになります。

しかし、この場合プログラムでBC=0を調べる必要はありません。実は、CP命令と違うもう１つのフラグ変化があるのです。それはP/Vフラグで、BCレジスタの値により次のように変化します。

BC≠0 の場合： P/V フラグ=0 (PO)
BC=0 の場合： P/V フラグ=1 (PE)

つまり、先ほどのプログラムは次のようにすることにより、正しく動作するようになるのです。

```
        LD      DE,0
CLOOP : CPIR
        JR      NZ,CPEND
        INC     DE
        JP      PO,CLOOP
CPEND :         :
```

この２つ（キャリーフラグとP/Vフラグ）以外は、「CP (HL)」と同様のフラグ変化をしますので、サインフラグによる条件分岐も可能です。

それにしても、いつの間にコンピュータの中に「正マ界」なんていう世界ができたのでしょうか……。



問その53 CPIR命令の効果的用法

ヤイ、ゲゲゲのQ太郎!! 勝手にウイルス妖怪退治なんて始めるのはやめてくれ。オイラのように正しいコンピュータの発展を願っている者には、はなはだ迷惑な話だぜ。だいたい、おまえはいつも正義面して厚かましいゾ!!

本当の正義の味方とは、オイラのことだ。ウイルス妖怪だって立派なプログラムなんだからな。コンピュータから見れば、有難いお客様だっていうことを忘れないでくれ。あんまりウイルス妖怪をいじめるなら、オイラがウイルスになって戦ってもいいんだぜ。

ところで、ゲゲゲのQ太郎の質問に答えたんだから、オイラの質問にもまじめに答えてくれよな。オイラのはQ太郎と違って、高級な質問だぜ。やはり、ある値があるかどうかを調べるんだが、チェックする値が複数なんだ。

たとえば、1000_H 番地から $1FFF_H$ 番地までにアスキーコードの 'A"D"F"K' のどれかが含まれているかどうか、それを調べようというんだ。

これには、CPI や CPIR 命令は使えないよな。やっぱり、CP 命令で1つひとつチェックするしか方法はないだろうな……。

でも、チェックする内容が多くなると、CP 命令というのは結構大変なんだぜ。オイラにもうまい方法を教えてくれ。

ドブねずみ男（邪マ界）

答

ゲゲゲのQ太郎氏から質問があった時、きっとこの手の人物（妖怪）からも質問が来ると思っていました。それが、まさかこんなに早くやって来るとは……。ア然!!

なにはともあれ、ここはまじめに答えなければなりますまい。もはや、誰が正義で誰が悪なのか、こちらにはサッパリわかりませんから。

まず、この質問の内容をそのままプログラム化してみましょう。

```
         LD    HL,1000H
         LD    BC,1000H
CLOOP :  LD    A,(HL)
         CP    'A'
         JR    Z,CPEND
         CP    'D'
         JR    Z,CPEND
```

```
        CP      'F'
        JR      Z,CPEND
        CP      'K'
        JR      Z,CPEND
        INC     HL
        DEC     BC
        LD      A,B
        OR      C
        JR      NZ,CLOOP
        RET
CPEND :         ⋮
```

内容の割には、プログラムが大がかりです。そこで、なぜCPIR命令が使えないという結論を出したのか考えてみましょう。これは、おそらく比較しようとする値('ADFK')がAレジスタになければCPIR命令は使えないと判断したためだと思います。

しかし、ここはコロンブスの卵で発想を転換したいところです。つまり、Aレジスタには1000_H番地からのメモリ内容を入れ、比較しようとする値('ADFK')をメモリ側に置くのです。文章ではわかりにくいので、裏レジスタを用いて実際にプログラムを組んでみます。

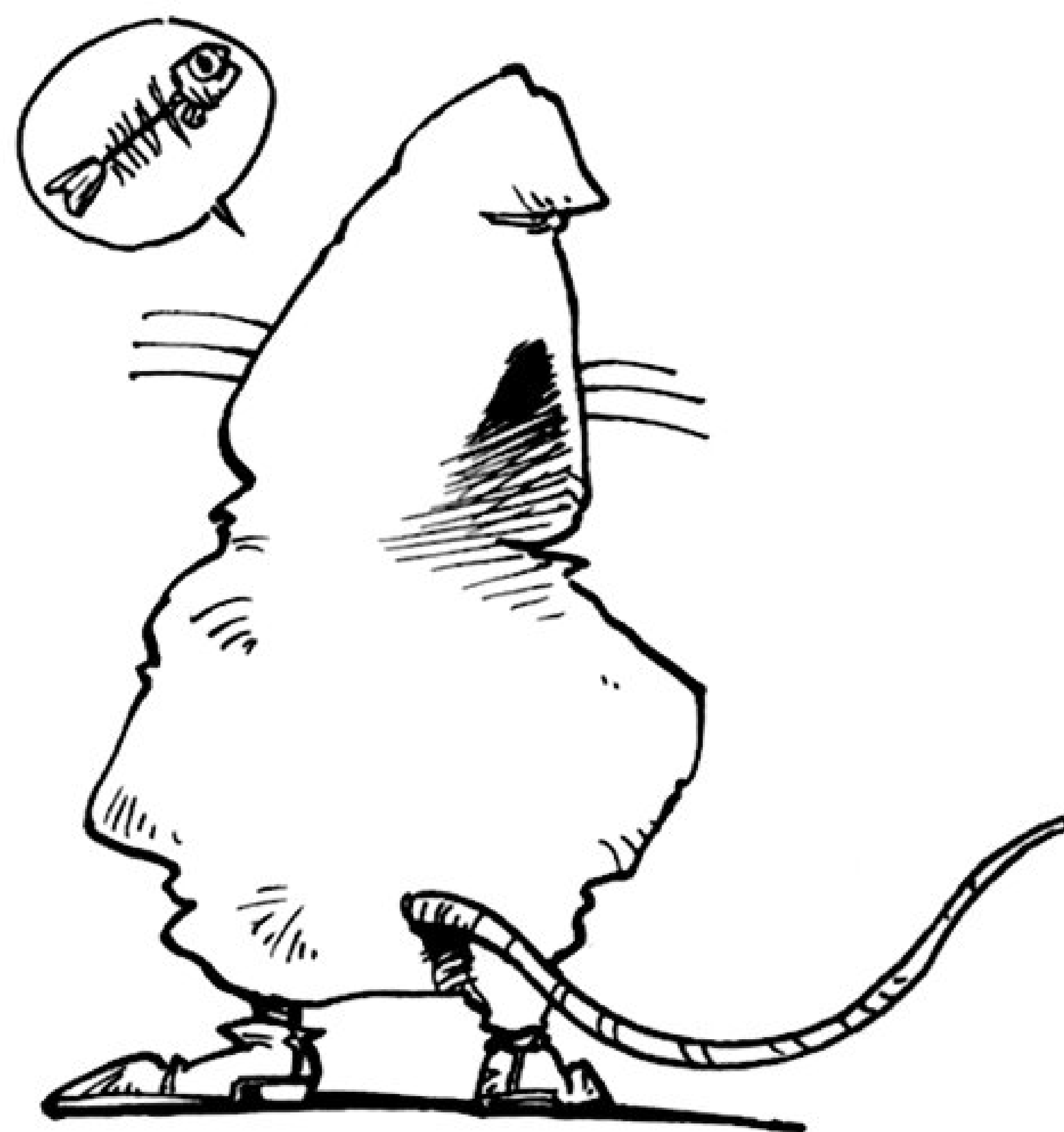
```
CDATA : DB      'ADFK'

        LD      HL,1000H
        LD      BC,1000H
CLOOP : LD      A,(HL)
        EXX
        LD      HL,CDATA
        LD      BC,4
        CPIR
        EXX
        JR      Z,CPEND
        INC     HL
        DEC     BC
        LD      A,B
        OR      C
```

```
        JR      NZ,CLOOP
        RET
CPEND :         ⋮
```

こうすれば、たとえチェックする値や順序を変更する場合でも、「CDATA」の内容や並びを変えるだけで簡単に変更できます。チェックする数が多くなればなるほど、このプログラムの簡略さが活きてくるでしょう。なお、EXX 命令ではAレジスタやフラグの変化はないという特徴を利用していることにも注目してください。

ドブねずみ男氏が、どのような目的でこの複数サーチ・プログラムを利用するのかわかりませんが、あまりコンピュータ内部でモメないようにお願いします。めったに出てこないウイルスより、身近な暴走のほうが困りますから……。



問 その54

CPDR命令とテーブル

コラ〜ッ!!　Q太郎にドブねずみ男。ケンカをするでにゃい。少しは、わしの立場も考えてくれにゃ、人間さまがコンピュータを嫌いになってしまうじゃろが。

わしか、わしはこの世界の長老'目ン玉おやじ'に決っておるじゃろ。初代コンピュータが巨大な箱としてこの世に生まれてから数十年、絶えることなくバグを繁殖させ、そしてそれを食べ続け、バグとともに生きてきたのじゃよ。

おまえらみたいに、ポッと出の妖怪とはレベルが違うからのう。ましてや、ウイルスなんぞはわしの歴史に比べたら、ひよっこもいいところじゃ。フォッフォッフォ……。

んにゃから、わしの質問はドブねずみ男なんか問題にならんほど高級じゃ。マ、少し似ておるといえないこともないがの。実は、コールすると押されたキーのアスキーコードを返す「KSCAN」というルーチンがあるんじゃ。その中にな、'A''D''F''K'のどれかが含まれていれば、それぞれに応じたルーチンへジャンプさせようというのにゃよ。

```
CLOOP :  CALL    KSCAN
         CP      'A'
         JR      Z,CCDTA
         CP      'D'
         JR      Z,CCDTD
         CP      'F'
         JR      Z,CCDTF
         CP      'K'
         JR      Z,CCDTK
         JR      CLOOP
          ⋮
```

これでは、さすがにCPIやCPIR命令は使えないじゃろ。こればかりは、CP命令で1つひとつジャンプさせるしか手はあるまい。じゃがな、チェックするキーの種類が多くなってくると、これも大変なことなんじゃ……。

目ン玉おやじ（本マ界）

答

とうとう出てきましたか。ウワサには聞いたことがありますが、まさか本物の'目ン玉おやじ'まで登場してくるとは!!

コンピュータの歴史はバグの歴史とも言われていますが、バグとは人間でい

うなら痛みのようなもの。もしも、人間が痛みを感じなければ、ケガをしようが骨折しようが、死んでも気がつかないという悲惨な状況になってしまいます。

こんな惨劇を未然に防ぐバグ……。ありがたいバグを育ててくれる'目ン玉おやじ'に心から感謝をしましょう。

さて、今回のようなケースは意外と多く出てきます。問（その24）にあったようなテーブルが利用できればいいのですが、チェックする内容が連続していないので、そのままでは利用できません。

チェックする数が少なければCP命令でもいいのですが、多くなることも考えて次のような手法があることも知っておくべきでしょう。

```
CDATA :  DB     'ADFK'
CTABL :  DW     CCDTA,CCDTD,CCDTF,CCDTK

CLOOP :  CALL   KSCAN
         LD     HL,CDATA+3
         LD     BC,4              ←BC＝チェックする総数
         CPDR
         JR     NZ,CLOOP
CCJMP :  SLA    C
         RL     B
         LD     HL,CTABL
         ADD    HL,BC
         LD     E,(HL)
         INC    HL
         LD     D,(HL)
         EX     DE,HL
         JP     (HL)
```

一見すると、CP命令でジャンプさせていた時より複雑になっているようですが、チェックするキーの内容やジャンプ先が一目でわかるので、デバッグ時などプログラムの内容を簡単に把握することができます。また、同じようなルーチンが多い場合は、「CCJMP」以下を共通ルーチンとして使うことができ、最終的には使用メモリ数も少なくなってくるでしょう。

このようなテクニックはゲームではあまり使うことはありませんが、各種のツールや実用ソフトにおいては結構有益なテクニックです。CPI(CPIR)、CPD

(CPDR) の活用法として覚えておくと便利です。なお、チェックする総数（＝テーブルの総数）が 128以下の場合、「CCJMP」中の「RL　B」は省略することができます。

プログラムにバグはつきものですから、結局はそれを速やかにリカバーできるようなプログラムがいいプログラムといえるわけです。



問その55 ベスト5のチェック

村おこし……といっても、雷おこしの親戚じゃありません。過疎化に悩む地域を活性化し、新しいエネルギーを発生させようという、すばらしい企画のことです。

今でこそ全国どこにでも見られる現象ですが、元祖はここ大分県の一村一品運動なのです。どんなことでも、最初に考えて実行するというのは大変なこと。マシン語のテクニックも同じですね。

そうとわかっていても、マシン語でオリジナルのテクニックを考えるなんて、まだまだ先の話です。とりあえず、現在のテクニックを覚えなければ……。ゲームセンターで、よくスコアのベスト５を表示しているのがありますが、あれはどうやるんでしょう。

たとえば、次のようにベスト５があるとします。そして、プレイした人の点数がHLレジスタにある場合、その人の成績が何番目であるか、ベスト５に入るならばベスト５の中に点数を組み入れるようにしたいのです。

```
SCOR1 :    DW    10000
SCOR2 :    DW     8000
SCOR3 :    DW     6000
SCOR4 :    DW     4000
SCOR5 :    DW     2000
```

わが村をマシン語村とするため、どうぞよろしく。

村長連合（大分）

答

他人のプログラムを評価するのが趣味のような人がいますが、そういう人に限ってオリジナルのテクニックは考えないものです。どんなにつまらないことでも、最初に考えるのはとても大変です。ましてや、それがブームになるということは、アイデア以上にその先見の明に感心してしまいます。

ゲームセンターで自分のスコアがベスト５に入るのは、名誉の表彰であり、また次にプレイするための刺激剤でもあるわけですが、これも最初に考えた人がどこかにいるはずです。ブームを越えて当然のように処理されているテクニック、これこそが究極のアイデアかもしれません。

この方法は色々あるでしょうが、結局は現在のベスト５と１つひとつ比較し

ていくことになります。次のプログラムは、HL レジスタに現在のスコアを入れてコールすると、ベスト５に入っていればそれを登録し、Aレジスタに順位（６位以下はすべて６）を入れて返すというものです。なお、ベスト５に同じスコアがあった場合は、新しいほうのスコアを上位に入れます。

```
SCORE :  LD     IX,SCOR1      ←SCOR1から2バイト単位でベスト5が登録されている
         LD     A,1           ←初期順位
         LD     B,5           ←登録スコア数
SCORL :  CALL   CPSCO
         DJNZ   SCORL
         RET

CPSCO :  LD     E,(IX+0)
         LD     D,(IX+1)
         OR     A
         SBC    HL,DE
         JR     C,CPSC1
         ADD    HL,DE
         PUSH   HL
CPSCL :  LD     (IX+0),L      ←新しいベストXを登録し、以前のベストX
         LD     (IX+1),H        以下のデータを繰り下げる
         EX     DE,HL
         INC    IX
         INC    IX
         LD     E,(IX+0)
         LD     D,(IX+1)
         DJNZ   CPSCL
         POP    HL
         POP    DE            ←SP合わせのためのダミー
         RET
CPSC1 :  ADD    HL,DE
         INC    IX
         INC    IX
         INC    A
         RET
```

では、ぜひマシン語で村おこしをお願いします。

問 その56

1バイト数値ソート

まろは徳川家の流れを祖母の父方の従兄弟（いとこ）の嫁の母の母方の父の祖父の兄の嫁の父の母方の叔父の叔母に持つという、由緒ある身分の生まれ……。

まろは高貴な育ちゆえ、幼少のころはバアヤが毒味をしたあとでなければ食事を許されていなかったのであるぞよ。おかげで、いまだに熱いものはダメ。まれにみる猫舌にてあるぞ。ところが、なんとラーメンが好きで好きでたまらないのよ。

さますとメンはノビる。さまさなければ熱い。このジレンマをなんとか解消したいのであるが、なにか妙手の心得はないかな……。

それはさておき、まろの職業は寺小屋（早い話が塾）の講師にあるぞ。まろは、そこにおいて家庭用電動頭脳（早い話がパソコン）を教えておるが、本当はよくわからんのであるよ。

先日も、子供たちに「マシン語でこの前のテスト結果を成績順に並べてヨ」と言われてしもうたが、難しいのでその日の講義はおひらきにした。100 点満点のテストを55人に実施した場合、バラバラになっている点数を成績順に並べるには、どんな方法でやるといいのかのう。

なお、子供たちの点数はメモリの「TENSU」に入っており、１人１バイトを使用しておる。まろの悩みを聞いてくれることを期待しておるぞよ。

徳ノ川秀麿（島根）

答

まず、猫舌のまろが苦心の末考えた秘伝のラーメンをお教えしましょう。その名は……'氷ラーメン'。できたてのラーメンに冷蔵庫の氷を４～５ケ入れるだけでOKです。真夏でも汗をかかずに、熱いラーメンを涼しく短時間で食べられるので、食後の満足感に壮快感も加わって、もう病みつきになること間違いなしです。ぜひお試しください。

満腹になったところで、数値ソートへと移りましょう。数値ソートとは、バラバラになっている数値を大きい順あるいは小さい順に並べ換えるものです。色々な方法が考えられますが、ここに紹介するのは小さい数値から順にフィックスしていくものです。

```
SORTS :   LD    C,55-1          ←C＝人数－１
SOTL0 :   LD    DE,TENSU
          LD    HL,TENSU+1
          LD    B,C
```

```
SOTL1 :   LD      A,(DE)          ←隣の数値と比較し、
          CP      (HL)              小さければ入れ換える
          JR      NC,SOTS1
          PUSH    AF
          LD      A,(HL)
          LD      (DE),A
          POP     AF
          LD      (HL),A
SOTS1 :   INC     DE
          INC     HL
          DJNZ    SOTL1
          DEC     C
          JR      NZ,SOTL0
          RET

TENSU :   DB      xx,xx, ……      ←55人分の点数が連続
```

プログラムの内容は、隣の数値と比較して小さければ入れ換えるという作業をC回繰り返すことにより、その中の最小値をまず求めます。次に、残された中から最小値を同じようにして求めます。これをC回繰り返すことで、小さい数値からメモリに固定されてソートが完了するわけです。

1バイトのソートは、色々なソートの基礎となるものですから、プログラムそのものよりもアルゴリズムを把握することが大切です。よくわからない場合は、紙切れに1～9までの数字を書き、バラバラにしてからソートするルールを考えながら並べ換えてみるといいでしょう。

最後に、全国のラーメン屋さんへ、全日本猫舌の会よりお願いです。メニューに‘氷ラーメン’を入れてください。

問 その57 2バイト数値ソート

初めてお便りを差し上げますが、私は田舎の中学校で教師をしている者です。昔は田舎の中学校といえば、オンボロ校舎に少ない生徒数というのが相場でしたが、最近は東京の地価高騰のあおりを受けて、都会風のハイカラな校舎に多数の生徒が通っています。

ただ、私のほうがオンボロ校舎タイプの教師なので、本当はこんな堅苦しい言葉が大の苦手です。でも、「……だっペ」とか「……ベェ」と言おうものなら、テレビの影響で標準語化した生徒に笑われてしまいます。実に、ツライ職業……。

そこで、せめて中身だけでも時代の先端を行くようにと、成績管理にパソコンを導入しました。プログラムにも、少しずつですがマシン語を取り入れています。とりあえずの処理として、テストの合計点数を成績順に並べたいのですが、2バイトのソートはレジスタをどのようにヤリクリしていいかわかりません。

次のように「TENSU」から 100人分の点数が2バイトずつ並んでいる場合、どのようにしてプログラム化したらいいでしょうか。

```
TENSU :  DW     358
         DW     125
         DW     532
          :
```

←100人分の点数が連続している

アー、標準語がツライ。どうやったらイイんだっぺ。

田舎教師（茨城）

答

オッ、懐かしいベーベー言葉。やはり、栃城（栃木と茨城）地方は、これが出てこなくちゃダメだっペナ。伝統の言葉を滅ぼそうとしているのは、やっぱりテレビの影響と東京の地価高騰だべ……。でも、例の語尾上がりのアクセントはまだ健在だっぺ。

それにしても、マシン語がいまだに機種によってバラバラなのは、なんと時代に逆行した現象なのでしょう。もしかすると、コンピュータの開発者というのはテレビを見ているヒマなどないのかもしれません。

とりあえず、本書はZ80地方の方言ですべてを表現しなければなりません。さっそく質問にある2バイトのソートを行うことにしましょう。

```
SORTS :   LD      C,100-1         ←C=トータル人数-1
SOTL0 :   LD      IX,TENSU
          LD      B,C
          LD      L,(IX+0)
          LD      H,(IX+1)
SOTL1 :   LD      E,(IX+2)
          LD      D,(IX+3)
          PUSH    HL
          OR      A
          SBC     HL,DE
          POP     HL
          EX      DE,HL
          JR      NC,SOTS1
          EX      DE,HL
          LD      (IX+0),E
          LD      (IX+1),D
          LD      (IX+2),L
          LD      (IX+3),H
SOTS1 :   INC     IX
          INC     IX
          DJNZ    SOTL1
          DEC     C
          JR      NZ,SOTL0
          RET
```

基本的なアルゴリズムは1バイトのソート（問その56）と同じなので、今回はプログラムがどのように変化したかを比べてください。

問 その58 ブロック単位の文字列サーチ

ビンボー、ビンボー、涙のビンボー。

この次は、どこの家にしようかな……。住みついて楽しいのは、モノスゴイお金持ちが坂道をコロげ落ちるように貧乏になった時だけど、いくら働いても貧乏という家も住み心地はいいもんだぜ。

だけどよ、お金っていうのは決してなくならないからな。ある所からお金が出れば、どこかにそのお金は入っていく。全員貧乏という理想郷は、実現できそうでなかなか難しいもんだわさ。そう考えると、あっしらの存在価値なんて、本当はあるのかないのかわからないんだよナァ。

それに最近は中流意識の強い家が多いから、どこへ住みついても大した感激はないもんだぜ。つまんないから、これからはコンピュータで適当な名前を作って、その名前があればそこに住みつくようにしようと思うんだ。

あっしらの専用ファイルに、8文字単位でインプットされた名字ファイルというのがあるんだ。たとえば、貧田さんという名字の家がある場合、そのファイルにはアスキーコードで「ヒンタ゛_ _ _ _ _」(_はスペースを意味する)と入っているんだ。

このファイルを 2000_H～$2FFF_H$ 番地にロードして、適当に作った名前があるかどうかをサーチしようというわけだ。内容は、そのサーチ・ルーチンをコールしたら、HLレジスタで名字の先頭アドレスを、そしてキャリーフラグで名字があったかどうかを返すようなもんがいいな。

おっと、そこから先のあっしの行動はまだ秘密だ。これは、いわゆる企業秘密ってやつだ。親切に教えないと次はおまえの家に住むからな。

貧乏神（貧民峡谷）

答

あれ……?? わが家にはすでに貧乏神が住みついていますよ。ワケもなく新しい貧乏神が来ると、古い貧乏神とケンカになりますぞ。なにしろ、うちの貧乏神様は先祖伝来の大物ですから……。

さて、これは文字列サーチの基本形となるプログラムですから、目的は別にしてグッドな質問といえるでしょう。

プログラムの使い方は、「SDATA」の部分にサーチしたい名字を入れ「SERCH」をコールすればOKです。

```
SDATA :   DB     'ヒンタ゛      '   ←名字（8文字）

SERCH :   LD     HL,2000H          ←サーチ先頭アドレス
          LD     BC,200H           ←サーチ名字総数（1000H/8）
SCHLP :   PUSH   BC
          CALL   CHEK8
          POP    BC
          DEC    BC
          LD     A,B
          OR     C
          RET    Z
          JR     SCHLP

CHEK8 :   LD     B,8
          LD     DE,SDATA
CHK8L :   LD     A,(DE)
          CP     (HL)
          JR     NZ,NOTSD
          INC    DE
          INC    HL
          DJNZ   CHK8L
          LD     BC,-8             ←8文字合えば、HLを名字のある先頭アドレス
          ADD    HL,BC               にする
          POP    AF                ←SP合わせのダミー
          POP    AF                ←SP合わせのダミー
          SCF                      ←名字があったことを示すフラグ
          RET
NOTSD :   LD     C,B               ←次のサーチアドレスにする
          LD     B,0
          ADD    HL,BC
          RET
```

プログラムを実行すると、サーチしたい名字があればキャリーフラグを立て、HLレジスタをその名字の先頭アドレスにしてもどります。

その後の処理は……、もうおまかせするしかありません。でも、うちに来ても餓死するだけですよ、きっと。

問その59 連続データからの文字列サーチ

なんみょうなみあむなむあみばてれんめんまちゃあしゅうねぎところこんぶなむなむかみさまほとけさまあーめんそーめんほうれんそうきつねにたぬきにてんたまおとしがまちゃんちゃがまがまがいものしんしゅんさんそんしゃんそんしょう……。

非常に読みにくいものを紹介しました。これは私が栄えある初代教祖となっている「ましむご教」の経典の頭の部分です。お経はこのような感じで8888文字からできており、信者は毎朝これを祈らなければなりません。

ところが、信者のみならずこの教祖である拙僧も、このお経をまだ全部は暗記できていないのです。なにしろ、深い意味がなさそうでありそうで、やっぱりないのがこの経典の特徴ですから、覚えるのは簡単ではありません。しかし、結果的にはこれも信仰心を高めることに役立っているのです。

というのは、「ましむご教」にはできるだけ速くお教を唱えた者が高い位につくという厳しい戒律があるからです。もちろん、教祖とて油断はできません。そのため、拙僧はこの大経典をコンピュータに入力し、毎日少しでも先まで覚えようとしています。そこで必要なのが、文字列のサーチです。

経典という長い文字列の中から、特定の文字列をサーチするにはどうするべきか。それができないようでは、教祖としての威厳もあったものではありません。なお、サーチする文字列の長さは不定ですが、最大で10文字程度です。

では、なむなむなむ……。

ましむご大僧正（魔心寺）

答

一度でいいから、なってみたいのが教祖さま。どんなにインチキくさい教祖でも、そこに信者がいるのが不思議です。どれほどのご加護があるのかわかりませんが、ましむご教も信者がいるから成立しているのでしょう。

それにしても、このお経と戒律。なんとまァ、ありがたさを感じさせない世紀末的ないい加減さ……。思わず入信したくなります……なんていう人がいるんでしょうか。

とにかく、ここは入信したつもりで希望通りのプログラムを組むことにしましょう。実行方法は「SDVAL」に文字列の文字数（≧１）を入れ、「SDATA」にサーチしたい文字列を入れて「SERCH」をコールします。

```
SDVAL :  DB     6                    ←文字数
SDATA :  DB     'ナムナムダ'    '    ←文字列

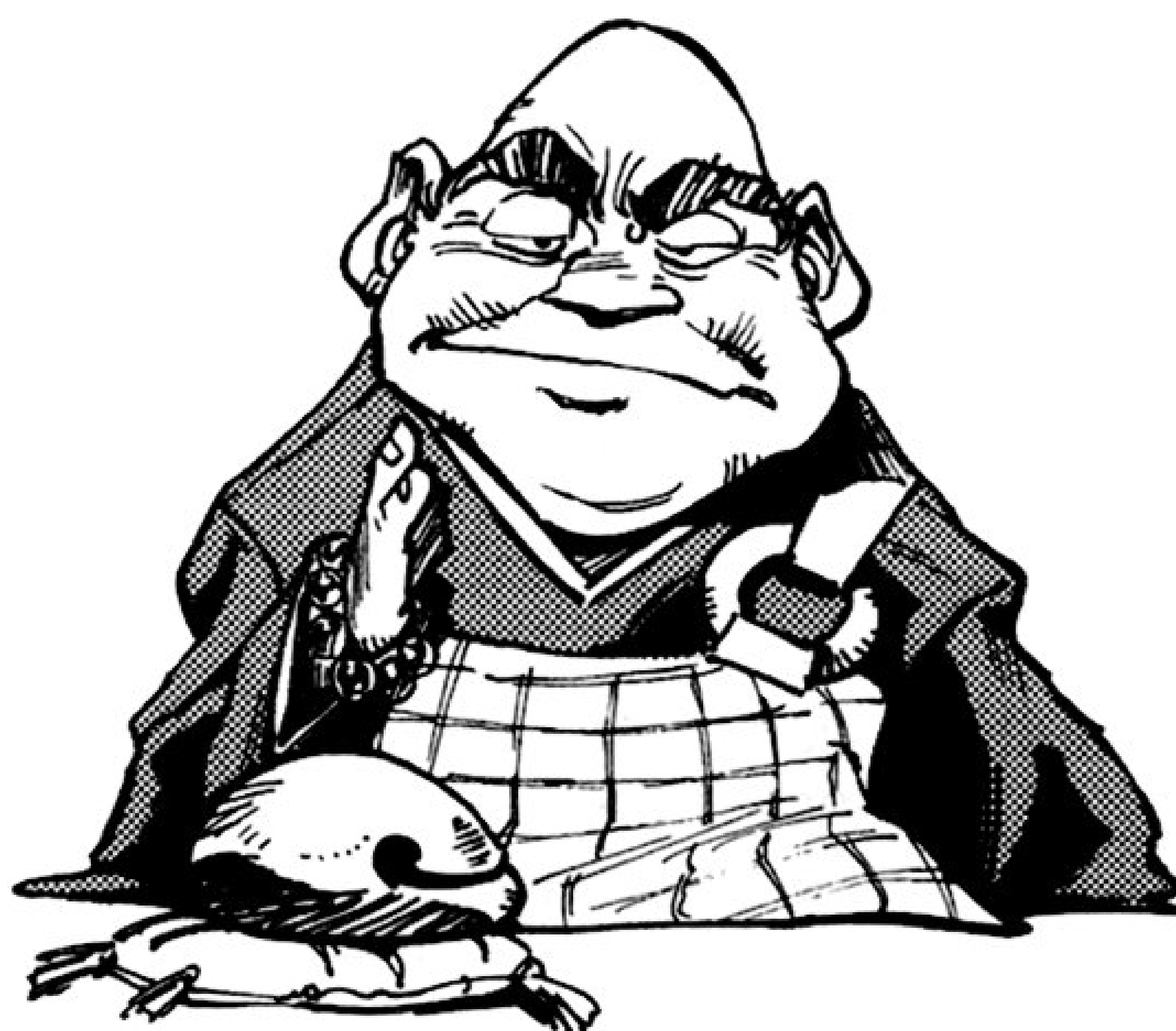
SERCH :  LD     HL,2000H             ←お経のある先頭アドレス
         LD     BC,8888              ←お経の総文字数
SCHLP :  LD     DE,SDATA
         LD     A,(DE)
         CPIR
         RET    NZ
         LD     A,(SDVAL)
         DEC    A
         JR     Z,SCHOK
         PUSH   HL
         PUSH   BC
         DEC    B
         INC    B                    ←B=0 でなければ BC≧100H
         LD     B,A
         JR     NZ,LENOK
         LD     A,C
         CP     B                    ←C≧文字列数−1の確認
         JR     NC,LENOK
         POP    BC
         POP    HL
         RET
LENOK :  INC    DE
         LD     A,(DE)
         CP     (HL)
         JR     NZ,NOTSM
         INC    HL
         DJNZ   LENOK
         POP    BC
         POP    HL
SCHOK :  DEC    HL
         RET
NOTSM :  POP    BC
         POP    HL
         JR     SCHLP
```

実行後、サーチしたい文字列が見つかれば、ゼロフラグを立ててもどります。HLレジスタは、その文字列のあった先頭アドレスを示しています。このプログ

ラムではサーチできる最大文字列数は10ですが、「SDATA」のワークエリアを多くすればいくらでも長い文字列のサーチが可能です。

なお、プログラムの細かな注意点として、サーチしたい最初の1文字目が見つかった場合、サーチ文字列数が1文字でなければ、お経の残り文字数がサーチする文字数分だけあるかどうかをまず確認しなければなりません。また、2文字目以降で一致しなかった場合には、次のサーチはその場所からでなく、本プログラムのように2文字目を改めて1文字目としてサーチし直さなければなりません。さもないと、サーチ文字列が重なるように含まれている場合（今回の例では「ナムナムナムタ゛」のような場合)、サーチ洩れを起こすことになります。

これで、教祖の威厳は保てるでしょう。なむなむなむ……。



問 その60 ブロック単位の文字列ソート

発明王トーマス・エジソン……この名を知らぬ者はおるまい。近代科学はすべてこのエジソンの発明があったからこそ生まれたのじゃ。エジソンがいなければ、未だにコンピュータなどなかったであろう。

わしはエジソンを神と崇拝し、エジソンに次ぐ発明を目指している町の発明家だ。本名を江路苫州（えじ・とます）という。もちろん、両親が付けてくれた本名だ。

町の者は、わしのことをエジソン、いや「えじさん」と呼んでおる。この響きが、わしにとってはたまらない魅力なのじゃ。

さてさて、わしだってエジサンの名に恥じないだけの膨大な発明がある。これまでに発明した名作数は 199にものぼっておる。特許や実用新案になったものは、残念ながらまだない。それが、町の発明家のいいところじゃ。ワハハハ……。

すべての発明には16文字以内の名がついておる。そこで、それらをコンピュータに登録してアイウエオ順に並べたいのだが、マシン語は発明ほどには得意でないのだ。

どうやっていいのか教えてくれたら、わしの発明した「灰皿パイプ」をプレゼントしよう。これは灰皿に長いチューブを付け、灰皿側のタバコをチューブでスパスパ吸うものじゃよ。ワハハハ……。

トマス・エジさん（富山）

答

ワハハハ……と言われても、タバコを吸わない人にとっては少しも役に立たないプレゼントです。せっかくですが、プレゼントは辞退しておきましょう。それより、発明の一覧表でも機会があったら見せてください。

ここに紹介するプログラムは、16文字ステップで文字列をソートするものです。原則的には同じ文字列がないことを前提としたプログラムですが、同じものがあってもソートに支障をきたすことはないでしょう。

発明の名前は 2000_H番地から16文字ごとに置いてあるものとします。したがって、たとえば「灰皿パイプ」であれば、「ハイサ゛ラハ゜イフ゜＿＿＿＿＿＿＿」(＿はスペースを意味する)のようにトータルで16文字になるように登録するということです。

また、ソート数（名前の総数）をCレジスタで扱っているため、ここでは最大ソート数は 256個 (0 の時 256) までです。これで足りない場合は、BCレジ

```
MSORT :  LD    C,199          ←登録してある名前の総数
SOTLP:   LD    HL,2000H       ←名前のある先頭アドレス
         LD    DE,2000H+16    ←2番目の名前のあるアドレス
         DEC   C
         RET   Z
         LD    B,C
SOTL1 :  PUSH  BC
         PUSH  DE
         PUSH  HL
         PUSH  HL
         PUSH  DE
SOTL2 :  LD    A,(DE)
         CP    (HL)
         INC   HL
         INC   DE
         JR    Z,SOTL2
         POP   DE
         POP   HL
         JR    NC,NOEXS
         LD    B,16
EXSOT :  LD    A,(DE)
         LD    C,(HL)
         LD    (HL),A
         LD    A,C
         LD    (DE),A
         INC   HL
         INC   DE
         DJNZ  EXSOT
NOEXS :  POP   HL
         LD    DE,16*2
         ADD   HL,DE
         POP   DE
         EX    DE,HL
         POP   BC
         DJNZ  SOTL1
         JR    SOTLP
```

スタをカウンタに使ってプログラムを変更してください。

ソートのアルゴリズムは、問(その56)や(その57)の数値ソートと同じです。アスキーコードも結局は1バイトの数値ですから、16桁の十六進数を数値ソートしたものと考えることもできます。

発明というと、つい特許大発明→大金持ちという発想をしてしまいますが、日本のエジさんはどうなんでしょうか。気になるところです……。

問 その61 A×B＝HL(標準タイプ)

超能力者……。信じられないかもしれませんが、ボクは超能力者です。テレビで有名なスプーン折りができるのです。

ただ、まだ超能力が弱いのでスプーンを1本折るのに3時間はかかります。それに、その間はスプーンを撫でながらズーッと念を入れなければならないので、とにかく非常に頭が疲れます。

そこで考えたのが、この念をパソコンにさせるということです。つまり、ボクがスプーンを折ろうとしている時に考えていることを、パソコンにそのままやらせればスプーンが折れるはずだと思うのです。

とりあえず、その準備プログラムとして必要なのが、A×Bの値をHLレジスタに入れるというかけ算プログラムです。現在は次のようにしています。

```
AXBHL :  LD      HL,0
         LD      D,0
         LD      E,A
AXBLP :  ADD     HL,DE
         DJNZ    AXBLP
         RET
```

かけ算としては平凡なプログラムと思いますが、もう少し速度を速める方法はないものでしょうか。なんとかして超能力パソコンを実現したいのです……。

エスパー魔脳（宮崎）

答

超能力者も大変そうです。そんなに苦労しなくとも、両手を使えばスプーン折りくらい簡単に実現できるのに……。

どうせ超能力を使うなら、スプーンやフォークを割箸のようにタテに割るとか、普通の人にできないことをやってもらいたいものです。どうして超能力者はスプーンの首ばかりを折ろうとするのでしょうか。

……という疑問は別にして、このパソコンによる超能力はすごく楽しみです。本当に成功してもらいたいものですが、その前にプログラムのバグは除いておきましょう。

このプログラムは、Bレジスタがゼロの時を無視しています。これでは、パ

ソコン超能力も失敗してしまいます。そういったバグも除いた上で、プログラムをもう少し高速化してみました。

```
AXBHL :    LD      HL,0
           DEC     B
           INC     B
           RET     Z
           LD      D,0
           LD      E,A
           CP      B
           JR      NC,AXBLP
           LD      E,B
           LD      B,A
AXBLP :    ADD     HL,DE
           DJNZ    AXBLP
           RET
```

高速化というより、計算効率を高めたというほうが適切な表現でしょう。最初にAとBを比較して、少ないほうをループする回数（かける数）にしただけですから、場合によっては（A≧Bの場合）まったく効果がないこともあります。

しかし、A＝1、B＝255 などの場合には絶大な効果がありますので、プログラムを組む際には当然考えておくべきテクニックといえます。

もし超能力パソコンが実現したら、ぜひそれでスプーンを縦に割ってください。首折りスプーンは、もう飽きましたので……。

問 その62 A×B＝HL(筆算タイプ)

「この石がいいね」と君が言ったから８月８日はゼハチ記念日
「16ビットになれよ」だなんて８ビット２個で言ってしまっていいの
キャリーというフラグをキープすることの期限が切れてポップエーエフ
ゼットハチマルでいいのというユーザーがいて言ってくれるじゃないのと思う
「Ａにロードしろよ」「ジャンプしろ」といつもいつも命令形でプログラムする君
⋮

わたしはマシン語を自在に使い、流れるようにプログラムする女流歌人です。自己紹介の代わりに、わたしの短歌作品群の中で特に気に入っているものを選んでみました。誰でもできる……と思っていただけるところに、この作品のよさがあります。

どこかで見たような短歌だと思う人もいるでしょう。実は、わたしもそう思っているのです。だから、オリジナルな短歌を目指して、マシン語プログラムのアイデアに磨きをかけています。

例えば、Ａ×Ｂ＝HL というかけ算を、通常わたしたちがするような形で計算できないものでしょうか。例えば、23×45を次のようにするやり方です。

```
     23
 ×   45
 ──────
    115  …… 23×5
    92   …… 23×4
 ──────
   1035
```

これができれば、プログラム効率はグンとよくなると思うのですが……。

マシン語歌人・俵まり（岡山）

答

マシン語を口語体のように使いこなし、流れるように美しいプログラムを組むというナゾの女流歌人。そんな歌人のマシン語短歌集『ゼハチ記念日』が 300 万部を超す売行きを示すという時代……になったらいいなと思うゼハチ記念日。

マシン語とはアルゴリズムの美を追求する言語です。足し算だけによるかけ算から、かけ算的なかけ算へとアルゴリズムを変えようとする点に、限られた文字の世界で美を追求する歌人ならではの美意識を感じます。

ただ、これをこのままプログラム化しても、ループによるかけ算がなくなる

わけではなく、単に計算効率をよくしたに過ぎません。また、一段目（115）と二段目（92）を加算する際には桁をずらさなければならず、思ったほどプログラムにはアルゴリズムの美が反映されないでしょう。

そこで、計算を次のように二進数で行うようにします。

```
         00010111   ……23
     ×   00101101   ……45
    ------------
         00010111
        00000000
       00010111
      00010111
     00000000
    00010111
   00000000
  00000000
  ------------
  000010000001011   ……1035
```

計算式の中にある二進数は、すべて 23×1または 23×0の結果ですから、ループによるかけ算をしなくて済むことになります。つまり、足し算のループから脱却できるようになるわけです。では、このアルゴリズムにしたがってプログラムを組んでみます。

```
AXBHL :   LD     HL,0        ← HL＝かけ算の結果
          LD     E,B
          LD     D,0         ← DE＝途中で加算される数
          LD     B,8
AXBL0 :   RRCA
          JR     NC,AXBL1
          ADD    HL,DE
AXBL1 :   SLA    E           ← DE の値を左シフト
          RL     D
          DJNZ   AXBL0
          RET
```

プログラムもスッキリしていますし、かけ算の内容によって所要時間（ステート数）が変わるということもありません。

こんなところにもマシン語の美が存在しているといえるでしょう。



問 その63

HL×BC＝HL(筆算タイプ)

アー、オホン!!

小生はバグというケチな虫でやんす。ご存じのように、小生は姿は見えねど自然発生的に生まれ、適当に暴れ回ったあとは強制的に抹消される運命でやんす。考えると、わびしい運命でやんすネェ。

その代わり、小生には駆除剤とかワクチンのような特効薬は存在しないので、プログラムさえあればいつでも生まれる可能性を持っているんでやんす。神様は、ちゃんと小生みたいな虫けらにも生きる道を与えてくれたんでやんすネ。

ところで、かけ算とは1バイト×1バイトとは限らないすね。計算結果が2バイトで収まりさえすれば、2バイト×2バイトのかけ算だって当然必要でやんすから。こんな場合は、どうやったらいいんでやんしょ。

足し算のループでやればいい……なんていう回答なら不要でやんすからね。速度とか美しさなんて小生はどうでもいいんでやんすが、小生の生まれる可能性が少ないプログラムは見てもつまんないんでやんすよ!!

ばぐちゃん（メモリの森）

答

プログラムのないところにバグは発生しませんが、バグもまた生きるためにプログラムを選んでいたとは気付きませんでした。眠たかったり疲れていたりすると、知らず知らずのうちにバグに狙われているのかもしれません。

しかし、2バイト（100_H以上）×2バイト（100_H以上）の計算は、結果が2バイトでは収まりませんから、正しい結果を求めようとすれば、どちらかは必ずFF_H以下でなければならないはずです。

このあたり、ついつい見逃すことがあるとバグのお世話になってしまいますから、注意が必要です。ただし、これはプログラムのバグというより使う側のミスですから、そのことが頭の中にあればバグとして問題になることはないでしょう。

では、HL×BC＝HLをプログラム化します。ついでですから、HL×A＝HLもプログラムにしておきます。

HL×BC=HL

```
AXBHL : EX    DE,HL
        LD    HL,0
        LD    A,16
AXBL0 : SRL   B
        RR    C
        JR    NC,AXBL1
        ADD   HL,DE
AXBL1 : SLA   E
        RL    D
        DEC   A
        JR    NZ,AXBL0
        RET
```

HL× A =HL

```
AXBHL : EX    DE,HL
        LD    HL,0
        LD    B,8
AXBL0 : RRCA
        JR    NC,AXBL1
        ADD   HL,DE
AXBL1 : SLA   E
        RL    D
        DJNZ  AXBL0
        RET
```

このプログラムでは、たとえ計算結果がオーバーフローを起こしても、オーバーした分が無視されるだけで暴走したりすることはありません。しかし、オーバーしたかどうかを示すサインはありませんから、利用する側でその可能性と必要があれば前もって値をチェックしなければなりません。

バグの発生する余地を残さないためには、この程度の気配りができるような健康状態を維持したいものです。



問 その64

A÷B＝C(小数第一位七捨八入)

水の都ベネチア……。いったい水の都とはどういう意味なんでしょう。水がおいしいのか、それとも川が多いのか、それとも島がたくさん浮かんでいるのか……。不思議な魅力に憧れて、ここベネチアへやって来ました。

まるで海の上に敷かれたような鉄道がベネチアへの入口です。列車の窓から見えるものは、右も左も広大な海原ばかり、いやがおうにも水の都への期待が高まります。やがて列車はサンタルチア駅へ到着です……。

駅前には広場があり歩道もあります。その向こう側には「TAXI」のマークが……。ところが、なんとこれが船なのです。タクシーが船ならバスも船。つまり、車道はすべて船道というわけです。

路地裏には歩道がなくても船道はあります。歩いていけない場所へも、船なら行けるのです。まさに水の都……。

そういえば、日本を出る時にマシン語でA÷B＝C（レジスタです）のプログラムをやりかけたままでした。気になるので、やっておいてください。ついでに、小数第一位で四捨五入するようなプログラムもお願いします。

旅情の人（ベネチア）

答

ベネチアは潮の香りでいっぱいです。百を超える島々が運河の道路を創り、運河は家と家を結び町を創ります。水の都というより水上都市という感じです。そんなところを旅している旅情の人……。実にうらやましい限りです。

さっそく、気になるA÷B＝Cをプログラム化してみましょう。一般に、割り算は引算のループによって実行されます。当然、B≠0 でなければなりません。

```
WARIS :   LD     C,0
WARIL :   SUB    B
          RET    C
          INC    C
          JR     WARIL
```

このプログラムでは、A－Bが何回実行できるかをCレジスタによってカウントしているわけです。キャリーフラグが立つまでループしますから、リター

ン後に「ADD　A,B」とすれば、Aレジスタには割り算の余りが求められます。

この余りが割る数（Bレジスタ）の半分以上であれば、Cレジスタを＋1することで小数第一位で四捨五入（十六進数なので正しくは七捨八入となる）をしたことになります。注意したいのは、Bレジスタを半分にして余りが出る場合は切り上げるということです（例えば7なら4とする）。

```
WARIS :   LD    C,0
WARIL :   SUB   B
          JR    C,WAEND
          INC   C
          JR    WARIL
WAEND :   ADD   A,B        ←A＝整数部での余り
          SRL   B          ←B＝B÷2
          SBC   A,B        ←七捨八入を実行
          RET   C
          INC   C
          RET
```

「SRL　B」による1/2では切り捨て（7なら3）となりますが、その時のキャリーフラグを含めて減算することで、結果的に切り上げた値を減算しています。

これで、安心して旅が続けられることでしょう。

問 その65 HL÷DE=BC(小数第一位七捨八入)

ニンニンニン……!!

拙者は伊賀の忍者。山を越え、野を越え、時代を越えて、現代社会へやって来たのでござる。得意の忍法は「排気ガス隠れ」……排気ガスに少し細工をして、煙玉で消えたようにするのでござる。

昔の忍者は修行に修行を重ね、必死になって独自の忍法を開発したのでござるが、すでに伊賀には『伊賀忍法トラの巻』があるでござる。それを見れば、こんな忍法なんて簡単なのでござるよ。

そんなことより、今は甲賀忍者のコンピュータに忍び込むほうが大変なのでござる。なにしろ、忍者のコンピュータというのはカラクリだらけ、まるで忍者屋敷のようなワナが待っているのでござる。

下手にプログラムを送ろうものなら、アッという間にやられてしまう。まずは敵の情報を分析しなければならぬでござる。そのためには、16ビットの割り算ができなければならない。それができないところに拙者の悩みがあるのでござる。

なんとか甲賀に内緒で教えてくれぬでござらぬか……。

影丸（三重）

答

伊賀の影丸……!? 懐かしい名前でござる。拙者は大の影丸ファンでござるゆえ、サインがほしいのでござるが……。

さて、16ビットの割り算ですが、問（その64）と同じような方法で実行させることができます。プログラムは、小数点以下切り捨ての場合と小数第一位で七捨八入した場合とに分けてあります。どちらも HL÷DE=BC という計算ですが、当然 DE≠0でなければなりません。

小数点以下切り捨て

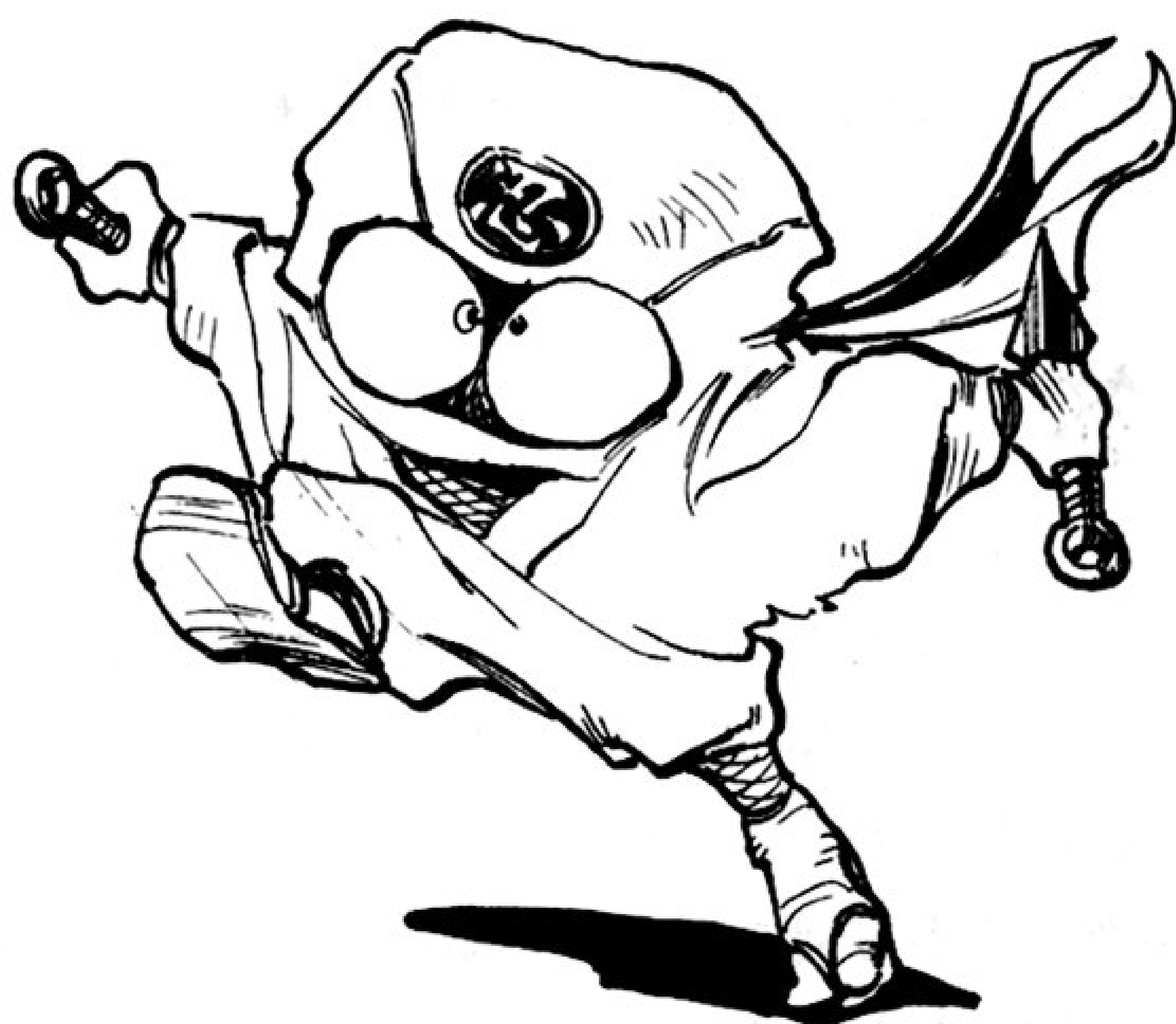
```
WARIW :  LD     BC,0
         OR     A
WARIL :  SBC    HL,DE
         RET    C
         INC    BC
         JR     WARIL
```

小数第一位七捨八入

```
WARIW :  LD     BC,0
         OR     A
WARIL :  SBC    HL,DE
         JR     C,WAEND
         INC    BC
         JR     WARIL
WAEND :  ADD    HL,DE
         SRL    D
```

```
RR      E
SBC     HL,DE
RET     C
INC     BC
RET
```

問（その64）を読まれた方には、簡単でござったであろう……ニンニン。



問 その66

HL÷DE＝BC.A(小数第三位七捨八入)

この伊賀者メ……!!　甲賀のコンピュータに無法侵入しようなんて、まだまだ考えが甘い甘い。拙者とて、おぬしの作るプログラムくらい見当はつくわ。

甲賀にも『甲賀忍法トラの巻』はあるし、伊賀のコンピュータに送り込む刺客プログラムも近いうちに完成するだろう。しかし、伊賀の忍者屋敷もなかなか手ごわいな。先日送った偵察プログラムは、とうとう戻ってこなかった。どうやら、敵のチェックプログラムに破れたようだ……無念。

そこで、拙者は伊賀の計算基準を超すような、正確な割り算プログラムで対抗しようと思うのだ。つまり、16ビットの割り算を小数第二位まで求めようというのだ。レジスタ構成としては、次のようなものを考えている。

HL÷DE＝BC.A

これまでは、商は整数だったから誤差がどうしても大きくなる。計算結果をDE倍しても元のHLとは相当違った値になる可能性を否定できなかった。これが小数第二位までになると、ほぼ正確に元の値に戻すことができるのだ。

(例) 四捨五入による計算
50÷30＝2　→　2×30＝60　……誤差が大きい
50÷30＝1.67　→1.67×30＝50.1　……誤差が小さい

これをなんとか実用化したい。伊賀に内緒で頼む。

サスケ（滋賀）

答

少年忍者サスケ……。拙者はサスケのファンでござる。ここに登場したサスケ氏はかなり大人びていますが、それでもやはりサスケはサスケ。ぜひサインをください。←なんと節操のない人物!!

それにしても、コンピュータをいじる影丸とサスケ……いったい現代の伊賀と甲賀の忍者は何を考えているのでしょうか。割り算がターゲットになる理由もさっぱりわかりません。とにかく、質問通りにプログラムを組むことにします。割る数(DE)のゼロチェックはしていませんから、その恐れがある場合は計算前にそのチェックをしてください。

```
WARIX : LD    BC,0
        OR    A
WARL1 : SBC   HL,DE              ← BC に整数部の商を求める
        JR    C,WAED1
        INC   BC
        JR    WARL1
WAED1 : ADD   HL,DE              ← HL=割り算の余り
        LD    (BCPOP+1),BC       ←整数部の商を退避
        LD    C,E
        LD    E,D
        XOR   A                  ← HL+00=HLA で表現
        LD    D,A                ← DE=EC とする (D=0)
        LD    B,A
WARL2 : SUB   C                  ← HLA÷EC の割り算
        PUSH  AF
        SBC   HL,DE
        JR    C,WAED2
        POP   AF
        INC   B                  ← B に小数部(小数第二位まで)の商を求める
        JR    WARL2
WAED2 : POP   AF
        ADC   HL,DE              ← HL=小数第二位まで割り算した余り
        SRL   D                  ← DE.C÷2
        RR    E
        RR    C
        SBC   A,C                ←小数第三位の七捨八入計算
        SBC   HL,DE
        LD    A,B                ← A=小数部の商
BCPOP : LD    BC,0               ← BC=整数部の商
        RET   C                  ←七捨八入
        INC   A
        RET   NZ
        INC   BC
        RET
```

整数部の商は簡単に求められますが、小数部はレジスタの役割とアルゴリズムを理解しておかないとわかりにくいかもしれません。順を追って確認しましょう。整数部の計算が終わった時点で、レジスタの内容は次のようになっています。

BC レジスタ＝商（整数部）

HL レジスタ＝割り算の余り

DE レジスタ＝割る数

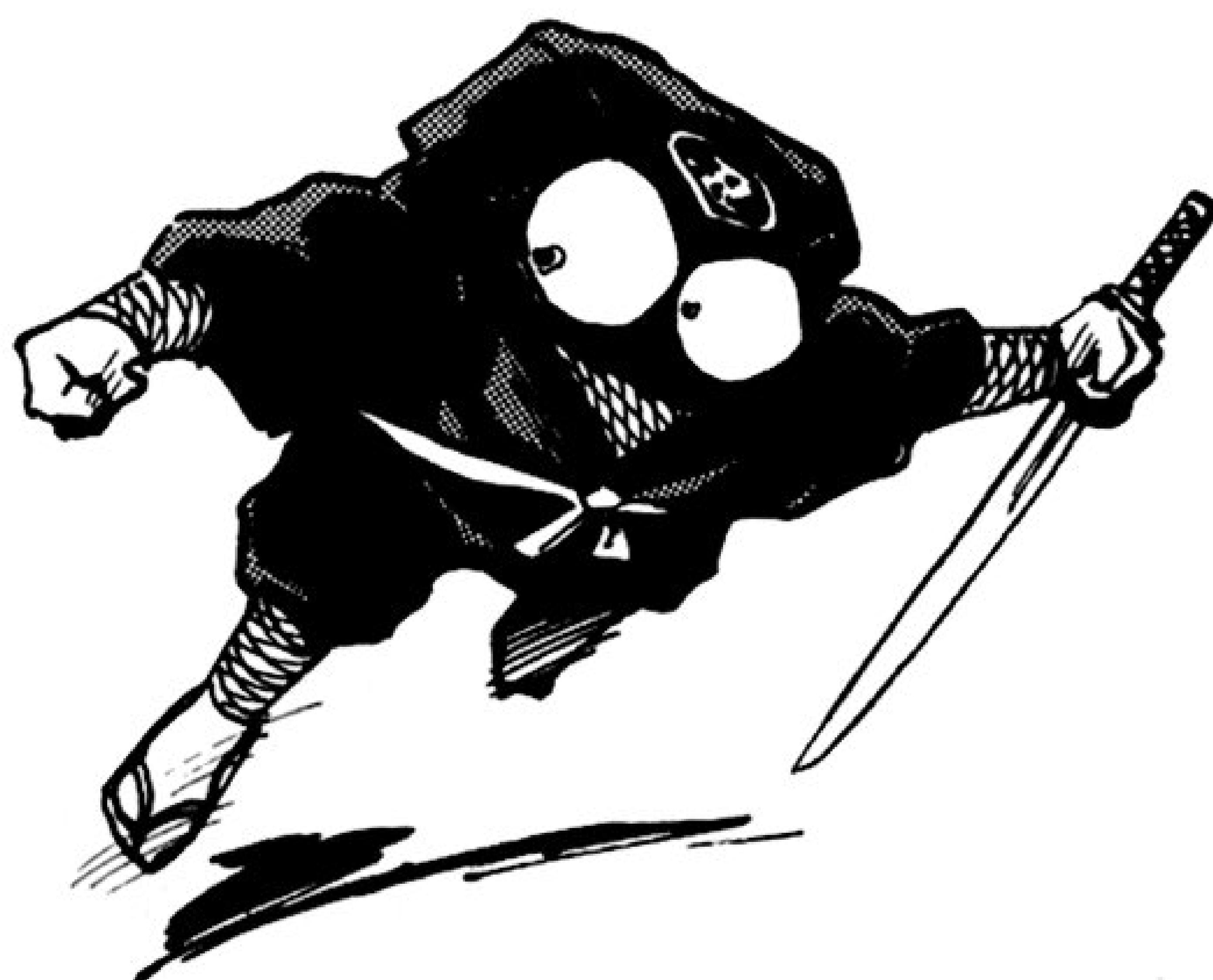
小数点以下の割り算は、HL レジスタを100_H倍して再び割り算をして求めています。感覚的には筆算の割り算と同じですが、単純に100_H倍すると HL レジスタがオーバーフローを起こす危険性があります。そこで、下 2 桁の'00'を A レジスタに入れると、HLA÷DE という感じの割り算になります。

これをそのままプログラムにすると、「A－E」「HL－D－CY」という減算ループをしなければなりませんが、「HL－D－CY」という命令はありません。そのため、プログラムでは、C←E、E←D、D←00とした上で、「A－C」「HL－DE－CY」の減算ループを実行しています。

なお、ここで求められる小数部の商は十六進数ですから、分数でいうなら 1／256 単位の数値となります。

最後に、「WAED2」から小数第三位の七捨八入処理をします。七捨八入した結果によってはAレジスタが桁上がり（FF_H→00_H）するかもしれませんので、その場合には整数部（BC レジスタ）が＋ 1 されるよう配慮しなければなりません。

忍法同様、最後の最後まで気をゆるめないでください……。



問 その67

B C.A×HL'=HL(小数第一位七捨八入)

その昔、アラブの砂漠で3人の男が遺産相続でモメていた。亡くなった父親の遺言によると「遺産のラクダは、長男が 1/2、次男が 1/4、三男が 1/6、仲良く分けろ」とあったそうだ。ところが、ラクダは11頭しかいない……。

今にもなぐり合いが始まろうという時、一人の老人が現れ「お若いの、わしの持っているラクダを一頭やるからケンカはやめるがいい」とラクダをくれた。三人は喜んで、12頭のラクダを遺言通りに分けた。

長男：12×1/2＝6
次男：12×1/4＝3
三男：12×1/6＝2

分けてみると一頭余っている。分配に満足した3人は、そのラクダを丁重に老人に返したそうだ。老人は、ニヤリとしていずこかへ去っていったという。

……こんな話をしながら、父がボクたちに 50,000円ものお年玉をくれるというのだ。ただし、長男が 270/555、次男が 180/555、三男が 105/555となるように、マシン語を使って分配するようにと命令された。

長男のボクとしては、なんとしてもこのプログラムを完成させなければならない。どうか、アラブの老人になってください。

総領の甚六（高知）

答

アラブの老人になってと言われても、あれは合計が 11/12だったからメデタシメデタシとなったわけで、ここで似たようなことをする気分になって 505円を出したとしても、結果は 555で割り切れるようになるだけでお金は戻ってこない……。

それなら、プログラムを組むほうがマシというもの。すでに、問（その66）によって小数第二位までの割り算ができますから、それをベースに各人の金額を求めてみることにしましょう。プログラムは、長男だけについて示しています。

```
MAN01 :  LD    HL,50000
         LD    DE,555
         CALL  WARIX       ← HL÷DE＝BC.A (問その66を利用)
         EXX
         LD    HL,270      ←裏 HL＝かける数
         EXX
         LD    E,A         ← BC.E＝かけられる数
         LD    HL,0
         XOR   A           ← HL.A＝かけ算の結果
         LD    D,16
AXBL0 :  EXX               ←二進数的手法によるかけ算
         SRL   H
         RR    L
         EXX
         JR    NC,AXBL1
         ADD   A,E
         ADC   HL,BC
AXBL1 :  SLA   E
         RL    C
         RL    B
         DEC   D
         JR    NZ,AXBL0
         CP    80H         ←最後の七捨八入
         RET   C
         INC   HL
         RET
```

割り算（50000／555）の結果に、長男の分として 270をかけるわけですが、かけ算の方法は問(その63)と同じです。しかし、小数点以下の処理が増えた分だけレジスタが不足します。そのため、かける数には裏レジスタを使用しています。

最終的な計算結果は、小数第一位を七捨八入したものが HL レジスタに入ります。A レジスタにある小数部の値はそのまま残っています。

《《《 計算結果 》》》

長男：24324　(HL：5F04H　A：42H)

次男：16216　(HL：3F58H　A：2CH)

三男：　9459　(HL：24F3H　A：6FH)

合計では 49,999円と割り切れなかった分の誤差が出ますが、小数点以下を考慮しない計算に比べればはるかに高精度です。もしも残った1円でモメるようであれば、最後の七捨八入を五捨六入(「CP　80H」→「CP　60H」)とすれば、三男の取り分だけが＋1されて合計でピタリ 50,000円になります。

現代版アラブの老人……。それは、コンピュータという砂漠に生きるマシン語プログラムのことかもしれません。



問 その68 HL÷DE＝BC.A（筆算タイプ）

『今日は東に明日は西……コピー求めて右往左往』

ここでいうコピーとは、宣伝文句という意味のコピーです。複写のほうのコピーではないので勘違いしないでください。冒頭のコピーは、そんなコピーライターの実体を文字で表現したものです。もちろん私のオリジナルコピーです。

カタカナ職業の花形ともいえるコピーライターですが、ハデな外見とは裏腹にハードな仕事です。文学的なセンスや広告の知識が要求されるだけでなく、落雷のようなひらめきが必要なのです。

そんなわけで、私はオンボロのキャンピングカーで‘ひらめき’を求めて日本国中をさまよっています。ただいま熊本あたりをさまよい中です。

そこで、気になるのが燃費……。これをマシン語で計算してみたいのです。それも、減算による割り算でなく、筆算のような感覚でやってみたいのです。

```
(例)      7.5
     80)600
        560
        ----
         40 0
         40 0
        -----
            0
```

どうでしょう。これはマシン語の‘ひらめき’ではありませんか。この方法でプログラムが可能になれば、あの非芸術的な減算ループ割り算から脱皮できるのでは……。

「HL÷DE＝BC.A」という割り算になれば最高です。

ピーコー物語（熊本）

答

マシン語も‘ひらめき’が勝負です。とはいえ‘ひらめき’とは稲妻……これが落雷するかカラ光に終わるかは、プログラミング技術にかかっています。このあたりのバランスはコピーライターに通じるものがあるかもしれません。

問（その62）で、かけ算を筆算的に行う方法の解説がありましたが、これはその割り算版といえるでしょう。かけ算が二進数で加算ループを回避できたように、割り算も二進数で行えば減算ループを回避することができます。なぜなら、二進数の割り算では商も二進数（1 か 0）となるからです。

（例） 600÷80＝7.5

```
                 0111.1   ←7.5
80→1010000 )1001011000.0   ←600
            0000000
            -------
            10010110
             1010000
            --------
             10001100
              1010000
             --------
               1111000
               1010000
               -------
                101000 0
                101000 0
                --------
                       0
```

これをプログラムにする場合、重要なのは割る数の桁数（例えば 1010000 で7桁）を知ることです。というのは、例からも明かなように、

商の桁数（計算回数）＝割られる数の桁数－割る数の桁数＋1

となるからです。ただし、この例のように商の頭に0がくることもあります。筆算による割り算では、これを無意識のうちに行っていますが、プログラム化する場合は商の桁数（計算回数）がわからないと、どこまで計算させていいのか判定できません。

また、マシン語では筆算のように部分的な減算ができませんから、割られる数が24桁（整数部16桁＋小数点以下8桁）あれば、割る数も24桁に桁合わせをしないとプログラムになりません。

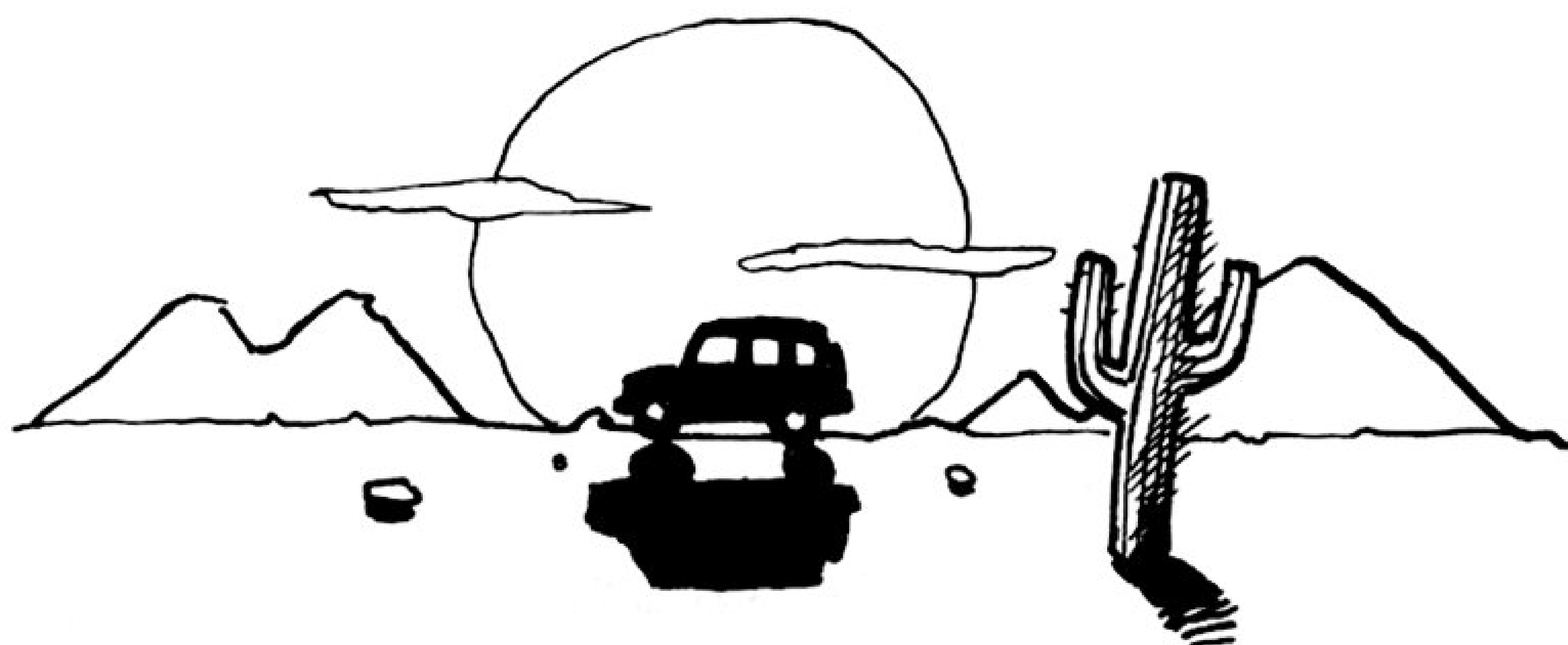
そこで、割る数の最上位ビットが1になるまで左シフトすれば、一段目の計算の桁合わせができると同時に、シフトした回数から割る数の桁数を求めることができます。二段目からは割る数を右シフトして計算の桁合わせをします。次のプログラムでシフトを多用しているのは、このような理由によるものです。

```
WARIZ :  LD     B,16           ←Bに割る数の桁数を求め、DEは最上位ビット
WAZL0 :  SLA    E                が1になるまでシフトする
         RL     D
         JR     C,WARZ1
         DJNZ   WAZL0
         RET                   ←DE=0 の場合は計算をしない
WARZ1 :  RR     D
         RR     E
         LD     A,25           ←A=整数部(16桁)+小数点以下(8桁)+1
         SUB    B
         LD     B,A            ←B=商の桁数 (計算回数)
         EXX
         LD     BC,0
         LD     E,0            ←裏レジスタのBC.Eが商となる
         EXX
         XOR    A              ←HL.Aが割られる数となる
         LD     C,A            ←DE.Cが割る数の初期値 (C=00)
WAZL1 :  LD     (KEEPH+1),HL
         SUB    C
         SBC    HL,DE          ←HL.A-DE.Cが成立すれば、キャリーフラグ
         CCF                     を立てる
         JR     C,WARZ2
KEEPH :  LD     HL,0
         ADD    A,C
         OR     A
WARZ2 :  EXX
         RL     E              ←裏レジスタのBC.Eをキャリーフラグを含め
         RL     C                て左へシフトする
         RL     B
         EXX
         SRL    D              ←DE.Cを次の段の計算に合わせ右へシフトする
         RR     E
         RR     C
         DJNZ   WAZL1          ←商の桁数だけ計算を繰り返す
         SBC    A,C            ←小数第三位の七捨八入処理。減算の結果、キャ
         SBC    HL,DE            リーフラグが立たなければ、小数第三位の
         EXX                     最上位ビット=1 (8H以上) となり小数第二
         LD     A,E              位を+1する
         PUSH   BC
         EXX
         POP    BC             ←BCが整数部の商
```

```
        CCF
        ADC     A,0            ←Aが小数部の商
        RET     NC
        INC     BC
        RET
```

こういった考えによる割り算では、小数点はユーザー側の目印にすぎません。したがって、整数商を求める割り算と考えれば、このプログラムは24ビット÷16ビットの計算をしたことになります。

減算ループによる割り算に比べ、メモリ消費も多いしプログラムも複雑ですが、商が大きくなればなるほどスピードアップ効果は大きくなります。割り算の目的と内容を考慮しながら、減算ループとうまく使い分けてください。



問 その69 アスキーコードからHLレジスタへ変換

西暦2100年……。

1900年代後半に出回っていたZ80は、すでにその役目を完全に果たし、CPUとして成すべき使命を全うしたかに思われていた。それどころか、その存在はコンピュータ史研究家の間でも忘れられた存在となっていた。

なにしろ、子供向けパソコンでさえ1024ビットCPUを8個搭載している時代だ。立体テレビによる完全3D画面が、まるでビデオ映像のようにパソコンで展開されている。しかも、それは家庭でのゲームである。ゲームセンターは亜空間体験ゲームルームとなり、プレイヤーは完全に立体虚像化された空間を自由にさまよったり、勇者として悪を倒すという古典的シナリオの世界で実際に戦ったりできるのであった。

私は、ドラゴラン銀河系ツバイシュタイン星タイムトラベラーである。地球の未来を見てきたので、正直に報告しておこう。

……で、私のタイムマシンであるが、メインCPUはなんとそのZ80なのである。ところが、どうもプログラムにバグがあるらしいのだ。アスキーコードで入力された年代（画面上の数字)を、マシン語プログラムで操作するために2バイトの数値に変換しなければならないのだが、それがうまくいっていないようだ。

ところが、残念ながら私はZ80マシン語はよくわからない。未来の情報を教えた代わりに、Z80でのプログラムを教えてもらいたい。

アストロ・ベイダー（TWS星）

答

テレビ画面の中で実際にプレイをする……これは、まさに究極のゲームといえるでしょう。もっとも、それが実現した時にはすでにそれ以上のゲーム欲が人間を支配しているのは間違いないでしょうが……。

平面型テレビの次は立体テレビとなると考えられていますが、いくら未来のテレビでも実像テレビ（テレビの中の料理が食べられる）だけは無理でしょうね。となると、虚像空間の次は何が出てくるのか、そのあたりの情報も知りたいものです。

未来はともかく、当面はアスキーコードで示された数字を2バイトの数値に変換するという現実的なプログラムを完成させなければなりません。これは、キースキャン結果がアスキーコードで返される場合や、テキスト画面上に表示されている数字を数値に変換するという場合にも必要な基本テクニックです。

```
ADATA :  DB      ' ','12345',' '     ←データはスペースで囲まれている

ATOHL :  LD      HL,ADATA+1          ← HL=データの先頭アドレス
         LD      A, ' '
         LD      BC,0FFFFH
         CPIR
         DEC     HL
         EX      DE,HL               ←DE=データのエンドサイン(' ')アドレス
         LD      HL,0                ← HL=十六進数に変換後の値
         LD      BC,1
         CALL    AREAD
         LD      BC,10
         CALL    AREAD
         LD      BC,100
         CALL    AREAD
         LD      BC,1000
         CALL    AREAD
         LD      BC,10000
         CALL    AREAD
         RET

AREAD :  DEC     DE
         LD      A,(DE)
         CP      ' '
         JR      Z,AREND
         SUB     '0'-1               ←アスキーコードを「数値+1」に変換
ARDLP :  DEC     A
         RET     Z
         ADD     HL,BC
         JR      ARDLP
AREND :  POP     AF                  ← SP 合わせのダミー
         RET
```

このプログラムでは、' '（スペース）をデータの前後に存在させることで数字の区切りとしていますが、これはケースバイケースで自由に変更することができます。また、変換できる最大数字数は5桁です。5桁以上の数字列の場合は後半の5桁が有効数字となります。

ただし、2バイト（0〜65535）を超えるかどうか、あるいはデータに数字以外の文字があるかどうか等、異常事態のチェックはしていません。キースキャ

ンデータの場合であれば、少なくとも数字の上下（'0'～'9'）くらいはチェックしたほうがいいでしょう。

これでタイムマシンが直ったなら、西暦2200年あたりのコンピュータ事情も調べてきてください。



問その70 アスキーコードからBCDの数値へ変換

こちらはツバイシュタイン星よりタイムマシンの製造依頼を受けたトライシュタイン星のトヨサン時動社です。実は当社のタイムマシンにバグが発見されたのですが、すでにツバイシュタイン星のタイムトラベラーは貴地球に向かってしまった後でした。

このままでは、時空間をフラフラする危険性があるので、ぜひ入力した年代をマシン語上で使用できるように修正したいのです。

プログラムの内容は、アスキーコードで入力された年代をBCDによる数値データに変換するというものです。もし、間違って2バイトの数値などに変換してしまうようなことがあると、ますます時空間の狂った世界へワープしてしまうでしょう。

ツバイシュタイン星の話では、乗務しているタイムトラベラーはマシン語が弱いとのことなので、おそらく地球にて誰かに質問するはずだということでした。きっとこの情報を受信できるような機関へ質問していると思われるので、そのような質問があったら次のように修正をするように連絡してください。

では、プログラムを送りままままままままーすすすす。ややややや、はははは発信機ががががが、ここここ故障ししししててててまま………。

技術部長ノホホン（TRS星）

答

運の悪い時はすべてがスレ違いになってしまうもの。タイムマシンのバグ発見が遅れたことで、運に見放されてしまったのかもしれません。ほんの少し前に、ツバイシュタイン星のタイムトラベラーから2バイトの数値に変換する質問があったばかりです。

こうなったら、運が好転することを期待するしかないでしょう。もしかすると、まだ今の年代をうろついている可能性もないとはいえません。とにかく、アスキーコードの数字列をBCDに変換するプログラム、これを急いで作ることにします。

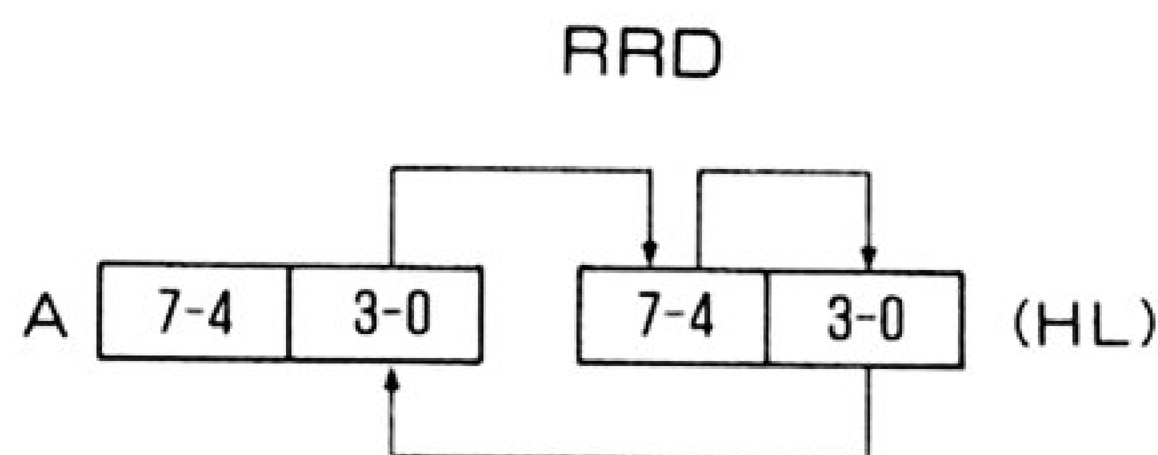
```
ADATA :  DB     ' ','12345',' '    ←データはスペースで囲む
BCDDT :  DB     0,0,0              ←BCDに変換された値が入る

ADBCD :  LD     HL,ADATA+1
         LD     A,' '
         LD     BC,0FFFFH
         CPIR
```

```
            DEC    HL
            EX     DE,HL                 ←DE=データのエンドサイン(' ')アドレス
            LD     HL,BCDDT−1            ←HL=BCD に変換された値が入るアドレス
            LD     B,3            ※       −1
ADBL0 :     INC    HL                    ← BCD データエリア・クリア
            LD     (HL),0
            DJNZ   ADBL0
            LD     B,3            ※     ←(HL=BCDDT+ 2 となっている)
ADBL1 :     DEC    DE
            LD     A,(DE)
            CP     ' '
            RET    Z
            RRD                          ←アスキーコードを BCD に変換
            DEC    DE
            LD     A,(DE)
            RRD                          ←アスキーコードを BCD に変換
            CP     ' '
            RET    Z
            DEC    HL
            DJNZ   ADBL1
            RET
```

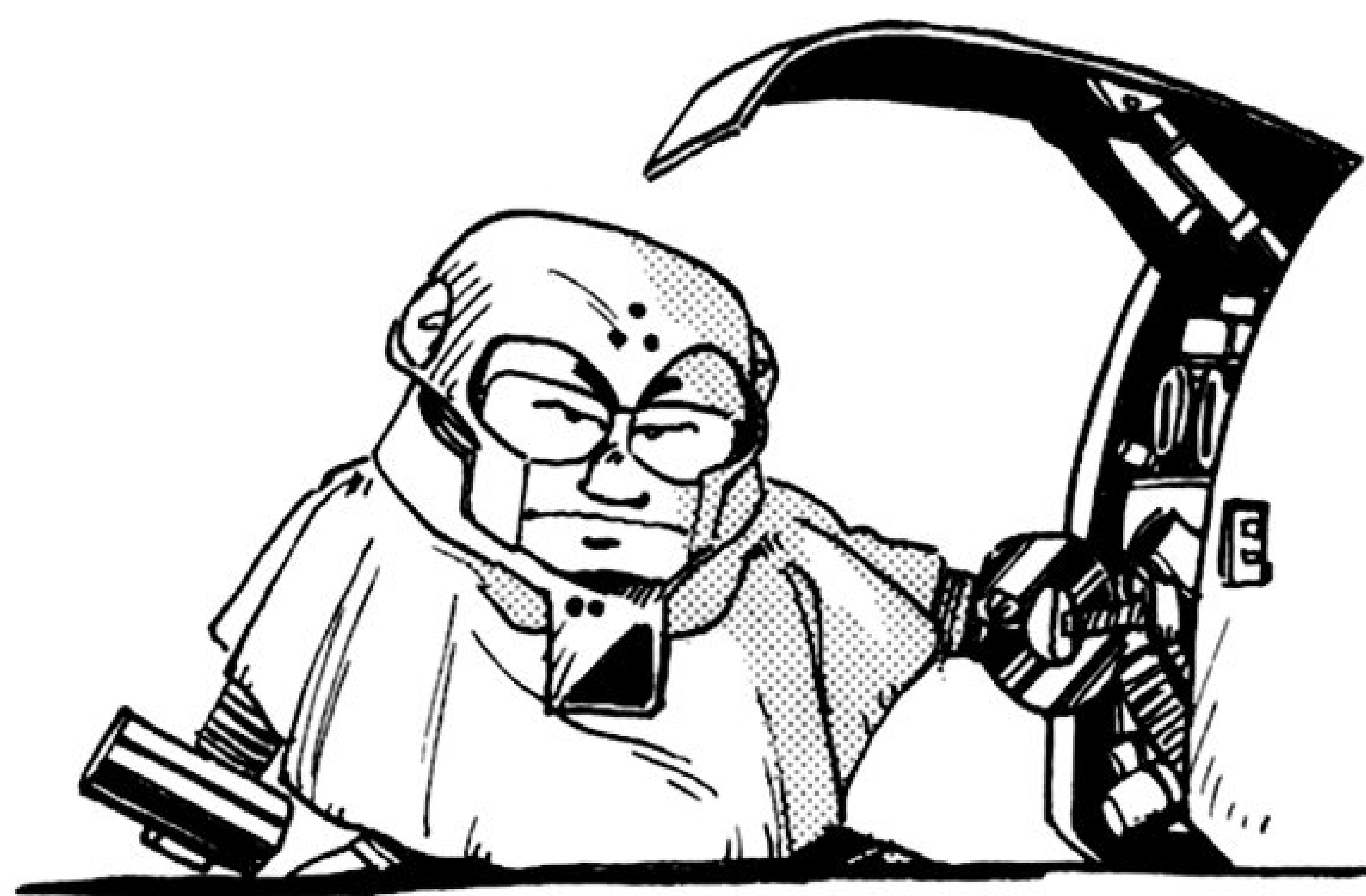
BCD に変換された数値は「BCDDT」に入ります。このプログラムではメモリを3バイトしか用意していませんが、メモリさえ確保すれば BCD による数値は桁の制限がありません。桁を増やす場合は、BCD のメモリ数を意味する B レジスタの値（※印 2 個所）も変更してください。

ここで、アスキーコードから BCD の数値に変換している RRD 命令に目を移してみましょう。これは下の図のような内容の命令ですが、アスキーコードの数字 (30_H〜39_H) は下位 4 ビットが BCD に必要な部分ですから、1 バイトにつき 2 度の RRD 命令を実行することで簡単に BCD の数値に変換することができるのです。



注意しなければならないのは、１度目のRRDでは(HL)の上位４ビットにデータが入るので、２度目のRRDを実行して初めて上位／下位の数値が正しくセットされるということです。そのため、本プログラムでは数字列の総数が奇数の場合には、ダミーとしてスペース(20_H)で２度目のRRD命令を実行してからBCD変換を終了させています。スペースの下位４ビット＝0であることが、このダミー処理を成立させています。

せっかく作ったプログラムですが、肝心の質問者は不明、連絡先も不明……。事情はよくわかりませんが、バグだけは今も未来も変わらぬ存在のようです。



問その71

BCDの数値をアスキーコードへ変換

ハイ、ちょっとそこのタイムマシン止まりなさい。ジグザグワープは銀河交通法で禁じられていますヨ……オイ、いつまでフラフラしたワープを繰り返すのだ。もしかして、酔っているのか。酒酔いワープは即１年間の免停と罰金20万ゴールドだぞ!!

アッ、また不法なワープをした。どこへ行くか、コラ……『銀河タイムパトロール隊員の白いタイムマシン』通称白タイの命令には絶対服従という法律を忘れたのか。この法律は各星が集まってできた銀河連邦の「時空ワープに関する交通法規特別準備委員会」によって正式決定された由緒ある基本六法の１つだぞ!!

オ、やっと止まった。なんだ、年代のアスキーコードをBCDに変換するプログラムを直していたのか。仕方ない、ジグザグワープの件は許そう。だが、ワープ先の年代はちゃんとタイムマシンの前後に表示しなきゃダメじゃないか。

その方法……？　それはBCDの数値をアスキーコードに変換して、指定のメモリに入れてやればいいのさ。ン、マシン語がわからない……!?　そんなムズかしいこと、オレに聞かないでくれよな。

だ、誰か……この会話を傍受していたらプログラムを組んでやってくれないか。そうしないと、オレも見張りとしてここを動けない……トホホ。

銀河のシェリフ（地球出身）

答

どうやらアスキーコードをBCDに変換するというプログラム（問その70）は届いたようですが、新たにその逆の問題が起きているようです。

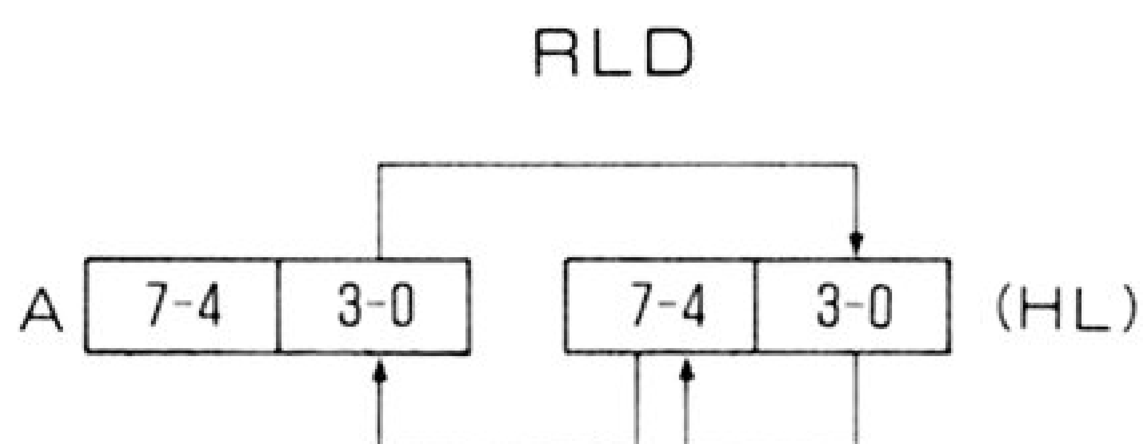
単純に考えれば、入力されたアスキーコードをそのまま使えばいいということになりますが、色々事情があって別のプログラムにしているのでしょう。このあたりはタイムマシンの設計者（本書の著者ではない）の意思を尊重するしかありません。

BCDの数値を上位／下位に分ける方法は問（その42）にもありましたが、プログラム的にもう少し簡単な方法があります。問（その70）にあったプログラムを逆用すればいいのです。

```
ADATA :    DB      0,0,0,0,0,0       ←アスキーコードに変換された結果が入る
BCDDT :    DB      12H,34H,56H       ← BCD による数値
```

```
BCDAS :  LD      HL,BCDDT     ← BCD による数値のあるアドレス
         LD      DE,ADATA     ←アスキーコード数字が入るアドレス
         LD      A,30H        ←30H='0'
         LD      B,3          ← BCDDT のバイト数
BCALP :  RLD                  ← BCD をアスキーコードに変換
         LD      (DE),A
         INC     DE
         RLD                  ← BCD をアスキーコードに変換
         LD      (DE),A
         INC     DE
         RLD                  ← (HL) の内容を元に戻す
         INC     HL
         DJNZ    BCALP
         RET
```

RLD による動作の内容は、次のようなものです。



Aレジスタの上位4ビットは不変ですから、最初にAレジスタを 30_H(アスキーコードの '0')としておけば、下位4ビットが入れ換わるたびにそのままアスキーコードの数字となります。

プログラムを見ると、1バイトのBCD数値を2つのアスキーコード数字に変換するために、RLDを3度実行しています。これは、動作の内容からもわかることですが、RLDは3度実行すると元の状態に戻るという特徴があるからです。当然、RLDを3度実行する前にAレジスタの値が破壊されると、「BCDDT」の内容は狂ってしまいます。

こういった特徴は、問(その70)で使用したRRDもまったく同じです。どのように使い分ければいいか、ちょうどLDI／LDDの違いにも似ていますので、これを機会に両者の性格をよく把握してください。手始めに、このプログラムをRRDを使用して作成し直してみるというのも一考です。

これで、白タイの隊員も無事拘束から解放されることでしょう。

問 その72 BCDをシフトする

オー、ワタシハ　ニホゴ　ヘタナ　ガイジ　デース。　チョドイマ　ニホノ　ゲンカン　ナリタヘ　ツイタバカリデース。

ワタシ　ウマレハ　フランス　ソダチハ　ブラジル→アメリカ→インド→ドイツ→サウジアラビア→オランダ→エチオピア→中国→ペルー→モンゴル→ケニア→日本→ポーランド→イタリア→ビルマ……。　ソノアトハ　キオクニ　アリマシェーン。

ダカラ　マトモニ　シャベレル　コトバ　ナーニモナイ。　セメテ　マシゴカンゼンニ　オボエタイネ。　ソコデ　シツモンスルアル。

レジスタノ　アタイヲ　2バイスルトキ　「SLA　A」トカ　シフトシマスネ。アレッテ　BCDニモ　ツウヨウシマスカ？

タトエバ、A＝5ヲ　2バイスル　プログラムハ　コレデ　OK　デスカ。

```
LD    A,5
SLA   A
DAA
```

ドゾ　ヨロシク　オネガイ　モシアゲマス。

ドコデモガイイジン（千葉）

答

ドモ　テイネナ　シツモン　アリガトゴザマス。　モシワケナイケド　フツノ　ニホゴデ　カカセテモライマス。

ふぅ〜っ……。カタカナに気を取られて、質問の内容を忘れるところでした。それにしても、さすがに多国を渡り歩いているだけあって質問も鋭い点を突いています。というのは、このプログラムは正解のようで正解でないし、間違っているけども結果は合っているという妙なものなのです。

まず、これを実行した結果はどうなるかというと、Aレジスタ＝10_H とキチンと2倍された値が得られます。したがって、ここでの値だけを見れば不正解とはいえません。しかし、これをA＝8_H で実行したらどうなるでしょうか。

結果がA＝16_H となってくれればいいのですが、実はこれもA＝10_H となってしまうのです。つまり、結果が正しくなる時もあるし狂う時もあるのです。

当然、これは間違った使い方ということになります。

DAA命令というのは、あくまでもBCD数値の加減算命令を実行した後にだけ正常な働きをする命令です。しかし、それ以外の場合でも無意識（？）にAレジスタの値をBCD化しようとする実直な命令なのです。その結果、偶然にも期待した値になることもあるし、まったく違った値になることもあるわけです。したがって、このプログラムは次のように改めなければなりません。

```
LD        A,5
ADD       A,A
DAA
```

ちなみに、左シフトして2倍になるとよく言いますが、これは正確には二進数を左シフトすると二進数表記で10倍（十進数表記で2倍）になるということです。つまり、何進数であれ左シフトすればその表記での10倍になっているのです。試しに我々が日常使用している十進数で考えてみましょう。

1→10→100→1000→10000→10000
5→50→500→5000→50000→50000

実に当り前のことですね。BCDというのは二進化十進数、つまり考え方としては十進数です。シフトさせるのであれば4ビット単位でシフトさせなければなりません。当然、結果は十進数で10倍となり、DAAをする必要はありません。参考までに、3バイトのBCD数値の左シフトを行ってみましょう。

```
DTBCD :   DB      00,12H,34H       ← BCDによる数値

SFT10 :   XOR     A
          HL,DTBCD+2
          LD      B,3              ← BCD数値のバイト数
LOOPS :   RLD
          DEC     HL
          DJNZ    LOOPS
          RET
```

プログラム実行後には、DTBCD＝01_H、23_H、40_Hとなります。RLD／RRD命令がシフト命令群の中にあることからも、シフトが二進数だけをベースにしたものでないことが確認できると思います。蛇足ですが、100 倍ならメモリ単位のブロック転送で処理できますし、右シフトをすれば結果は 1/10 となります。

デハ　ドゾ　マシゴデ　セカイノコクサイジンニ　ナテクダサ〜イ……。



問 その73　BCD数値の四捨五入

ウワッ!!

また仲間がやられてしまった。今度の相手はとても強い……。だいたい、アイツら4人組だもんナァ。魔法を使うヤツもいるし、ケガを治すヤツもいる。おまけに戦うたびにますます強くなっていくようだ。

これじゃ、いくらこちらが頑張ってもかなうわけないよナ。おや、マタンゴおやじがイイ線いってるぞ。うまいことアイツらを眠らせたようだ。そこだ、行けっ!!　なぐれ、パンチだパンチ!!　そうだ、ボディ……ボディ!!

やったァ〜。全員倒したぞ〜。これで、アイツらの顔は二度と見ることはない。平和な世界が戻った……トト、と思ったらアイツら生き返っちまった。

なんてイイ加減な世界なんだ。ここはデジタルの世界のはずなのに、まるで四捨五入の世界みたいだ。

そういや、BCDの数値を四捨五入するってのはできるんだろうか。たとえば、メモリを3バイト（6桁）使って、前半の5桁を整数部、残り1桁を小数部とし、小数第一位四捨五入なんていうのはできるんだろうか。

どうせ、オレたちの存在なんてゴキブリ以下なんだろうけど、それくらいはハッキリしておきたいよナ。アイツらいくらでも生き返れるからいいけど、オレたち死んだらしまいのワビしい人生なんだからサ。

スライムちゃん（テレビ界）

答

グチを言いたいのか質問をしたいのか、本音は不明ですが、勇者の一人としてスライムちゃんの気持ちがわからないわけでもありません。かなりハデに暴れ回った経験がある以上、せめてもの罪ほろぼしに、無条件でBCDの四捨五入を教えることにしましょう。

四捨五入とは、4以下切り捨て5以上切り上げですから、＋5をして9以下切り捨てと考えることができます。ですから、小数第一位四捨五入なら、0.5を足して小数点以下を切り捨てればいいのです。

〈〈〈 小数第一位四捨五入の例 〉〉〉

12.3→12.3＋0.5＝12.8→12.0

12.7→12.7＋0.5＝13.2→13.0

では、質問にあるように3バイト(6桁)のBCD数値の1桁目で四捨五入するプログラムを組んでみます。

```
DTBCD :   DB      12H,34H,56H         ←BCDデータ

SISYA :   LD      HL,DTBCD+2
          LD      A,(HL)
          ADD     A,5                 ←四捨五入のため+5する
          DAA
          PUSH    AF                  ←キャリーフラグを退避
          AND     11110000B           ←1桁目をゼロにする
          LD      (HL),A
          DEC     HL
          POP     AF
          LD      A,(HL)              ←以下、通常のBCD計算
          ADC     A,0
          DAA
          LD      (HL),A
          DEC     HL
          LD      A,(HL)
          ADC     A,0
          DAA
          LD      (HL),A
          RET
```

四捨五入するのは1桁目ですが、桁上がりがあるかもしれませんから計算は全桁に渡って行わなければなりません。

ちなみに、単なる切り上げは零捨一入のことですから、+9をして切り捨てをすればいいわけです。もちろん、二捨三入でも八捨九入でも自由自在です。いかにも、スライムちゃんにふさわしいプログラムではないですか……。

問その74 BCD数値×A

先日、友人と３人でひなびた民宿へ泊まった。一人一泊1,000円という安さだった。次の日、ぼくは友人２人から1,000円ずつ集めて、まとめてお金を払おうとした。すると、バァさんが「また来ておくれ」と言って 500円サービスしてくれた。三人で 500円じゃ割り切れないので、ぼくはこっそり 200円ネコババした。そして、残りをみんなで 100円ずつ分けた。

結局、一人 900円で泊まったわけだから、三人で2,700円だ。それに、ぼくのネコババした200円を足すと……。アレッ、2,900円しかない。確か、最初は3,000円あったはずだ。どこかに落としたのだろうか。

ぼくは、コンピュータでこの謎を解決しようと思う。数値はBCDで３バイトを使うことにした。そうだ、色々な人数でも通用するようにしたほうがいい。ぼくは、人数はそれほど多くならないからＡレジスタで示すことにした。だから、Ａレジスタは十六進数の数値ということになる。

ということは、『３バイトのBCDの数値×Ａ』ができればいいのだな。ぼくは、いつものように独り言をつぶやきながら、プログラムを組むことにした。でも、BCDのかけ算ができなかった……。

セブンっ子（鳥取）

答

数のマジックという言葉がありますが、とかく人間は数にダマされやすいものです。朝三暮四とは猿のこと、なんて思っていると痛い目に会うかもしれません。

８畳の広い和室	←畳そのものが極端に小さい
３個で９割引	←３割引×３個
定価10,000円を980円	←500円くらいの商品に10,000円の定価を付ける
合格率100％の予備校	←１人が５校に合格すると４人が不合格でも合格率は100％

時ソバ、ねずみ講、減税と増税……、わかっていてもついついダマされそうです。もっとも、今回のネコババの計算は自業自得ですが……。

ネコババ計算の解決は別として、『３バイトのBCDの数値×Ａ』という計算

は正しくプログラムしなければなりません。ここでは、Aレジスタの値がBCDではなく十六進数として扱われているので、足し算のループでかけ算を実行します。

```
KEKKA :  DB     0,0,0               ←かけ算の結果が入る
DTBCD :  DB     01H,23H,45H         ← BCD の数値

KAKBA :  LD     HL,KEKKA            ←結果エリアのクリア
         LD     B,3
KBAL0 :  LD     (HL),0
         INC    HL
         DJNZ   KBAL0
         LD     B,A                 ← B=ループ回数（かける数）
         OR     A
         RET    Z
         LD     HL,DTBCD
         LD     C,(HL)              ← C=(DTBCD)
         INC    HL
         LD     E,(HL)              ← E=(DTBCD+1)
         INC    HL
         LD     D,(HL)              ← D=(DTBCD+2)
KBAL1 :  LD     HL,KEKKA+2          ←加算ループによるかけ算
         LD     A,(HL)
         ADD    A,D
         DAA
         LD     (HL),A
         DEC    HL
         LD     A,(HL)
         ADC    A,E
         DAA
         LD     (HL),A
         DEC    HL
         LD     A,(HL)
         ADC    A,C
         DAA
         LD     (HL),A
         DJNZ   KBAL1
         RET
```

かけ算の結果は「 KEKKA」からの3バイトに入ります。BCDによる最も単純な計算例ですから、数のマジックでゴマ化されることもないでしょう。

問その75 BCDどうしのかけ算(筆算タイプ)

からっ風とカカア天下……とくりゃ、ウチの母ちゃんにピッタリ。暑さ寒さもなんのその、強くたくましく頼りになる母ちゃんだ。

北風がビュウビュウ吹いたって、絶対に倒れないドッシリした体格。ワシが安心して働けるのも、この母ちゃんがいるからだ。給料が安くても文句は言わない。おまけに、料理はうまいし運転もできる。

母ちゃんが病気になってもワシは抱えられないが、ワシが病気になると母ちゃんはヒョイと抱えてくれるんだぜ。どうだい……カカア天下はいいだろう!!

でもな、いくら母ちゃんでもコンピュータだけはダメだ。これだけは教えてもらうことができない。仕方ないので質問だ。

実は、BCDでかけ算をやってみたいのだ。もちろん、問(その62)にあるようなカッコイイ方法でだ。全部を足し算のループでグルグル回すなんていうのはダメだぞ。なにしろワシはソフトハウスの部長という肩書だからな。

おっと、レベルの低いソフトハウスだなんて思わないでもらいたい。ワシは営業部長だからプログラムは関係ないんだ。これはあくまでワシの趣味だ、し・ゅ・み……。

信じられないんだったら、母ちゃんに聞いてみな。

ワシは父ちゃん(群馬)

答

なになに……「追伸　書き忘れたけど、母ちゃんは結構美人なんだぞ……」だって。こうなると、質問というより母ちゃんの宣伝みたいな気もします。

なにはともあれ、すべてを信じてBCDのかけ算をカッコヨクやることにしましょう。ただし、問(その62)の場合のように二進数のかけ算としてプログラムを組むことはできません。なぜなら、BCDの数値をシフトすると、BCD演算に不適な(十六進数でA～Fを含む)数値となることがあるからです。

問(その72)でも説明したように、BCDでは数値を4ビット単位で扱うことが原則です。したがって、BCDのかけ算は筆算による十進数のかけ算と同じような感覚でプログラムを組むことになります。なお、その際に行う桁単位のかけ算は、足し算のループで行うしかありません。

<<< プログラム的筆算によるかけ算の例 >>>

```
  123
×456
 ────
  738 ←123×6  (足し算のループで行う)
 6150 ←1230×5 (足し算のループで行う)
 492  ←123×4  (足し算のループで行う)
 ────
56088
```

では、これを実際にプログラミングしてみましょう。このプログラムでは、ADATA（3バイト：十進数で6桁）× BDATA（3バイト：十進数で6桁）= KEKKA（6バイト：十進数で12桁）の計算を行っています。

```
KEKKA :   DB     0,0,0,0,0,0
DUMMY :   DB     0              ←ADATAを4バイトとして扱うためのダミー
ADATA :   DB     0,01H,23H
BDATA :   DB     0,04H,56H
ADA10 :   DB     0,0,0,0        ← ADATA×10が入る

KAKEX :   LD     HL,KEKKA       ←結果をクリア
          XOR    A
          LD     B,6
KKIL0 :   LD     (HL),A
          INC    HL
          DJNZ   KKIL0
          LD     HL,ADATA       ← ADATA×10をADA10に用意する
          LD     DE,ADA10+1
          LD     BC,3
          LDIR
          EX     DE,HL
          LD     B,4
KKIL1 :   DEC    HL
          RLD
          DJNZ   KKIL1
          LD     IX,BDATA+2     ← IX=かける数のエンドアドレス
          LD     HL,KEKKA+5     ←(KEKKA)~(KEKKA+5)に計算結果が入る
          LD     C,3            ←かけるバイト数
KAKLP :   LD     DE,ADATA+2
          LD     A,(IX+0)
          PUSH   AF
          CALL   KAKSS
```

```
        POP     AF
        RRCA
        RRCA
        RRCA
        RRCA
        LD      DE,ADA10+3
        CALL    KAKSS
        DEC     IX
        DEC     HL
        DEC     C
        JR      NZ,KAKLP
        RET

KAKSS : AND     00001111B          ←桁ごとのかけ算
        RET     Z
        LD      B,A
KAKSL : PUSH    HL
        PUSH    DE
        LD      A,(DE)
        ADD     A,(HL)
        DAA
        LD      (HL),A
        DEC     HL
        DEC     DE
        LD      A,(DE)
        ADC     A,(HL)
        DAA
        LD      (HL),A
        DEC     HL
        DEC     DE
        LD      A,(DE)
        ADC     A,(HL)
        DAA
        LD      (HL),A
        DEC     HL
        DEC     DE
        LD      A,(DE)
        ADC     A,(HL)
        DAA
        LD      (HL),A
        POP     DE
        POP     HL
        DJNZ    KAKSL
        RET
```

桁ごとのかけ算といっても、1バイトが2桁分に相当していますから、すべての桁を同じように扱うわけにはいきません。先ほどの例でいうと、普通の筆算では2段目は 615となりますが、1段目とは「738＋6150」という計算です。つまり、2段目の値は10倍して加算しなければならないわけです。もちろん、3段目は 100倍して加算することになりますが、これはメモリを1バイトずらすことで対処できます。しかし、次の段があればやはり10倍した値が必要になります。

そこで、最初にADATA（かけられる数）の値を10倍したものを用意しておくと、かける数の上位4ビットも下位4ビットと同じような計算処理ができます。ただし、10倍したものは4バイトのメモリを使いますから、プログラムは4バイト分の計算（KAKSL内における処理）をしなければなりません。さらに、このプログラムを上位／下位共通に使用するためには、ADATAのほうも4バイトにしておく必要があります。そのため、ADATAの手前にDUMMYとして1バイト（中身＝0）を確保しているわけです。

プログラム(KAKSS)を共通化せず、下位4ビット用に3バイト分の計算をするプログラムを用意すれば、このDUMMYは不要です。このあたりの判断は、営業部長の父ちゃんにまかせることにします。

問 その76

HLレジスタの値をBCDに変換

ピラミッド・パワー……。この謎のベールに包まれたパワーを確認するため、私はクフ王の眠る巨大なピラミッドへとやって来ました。

私は、すでに30年の歳月をかけて、この神秘のパワーを求める物理学的計算式を発見したのです。それは、あらゆるパワーが複雑に、しかもバランスよく絡み合った、まるで生命体のような計算式でした。

しかし、その計算式には1つだけ未知数が存在しているのです。これは、現在の数学では発見されていない数値、たとえていうならi（二乗して−1になる数値）のようなものです。私は、その未知数にPp（ピューピー）と名付け、この計算式を成立させたのです。

では、このPpの正体だけを明かしておきましょう。Ppとは、かけても割っても1になるという数値です。例をあげてみます。

543×Pp＝1　　543÷Pp＝1

つまり、あらゆる数値が1に収束してしまうのです。この未知数Ppの存在を裏付けるため、ピラミッド内部のアチコチでPpの測定をしています。その時、マシン語でBCDの数値と十六進数のHLレジスタの値との高速乗算をしたいのです。

ピラミッド内部より、絶大なるご協力を要請します。

デタラム・ペテム（エジプト）

答

ピラミッドだけでなく、単なる四角錐にも謎のパワーが秘められているそうですが、そのパワーとは一体何なんでしょうか。卵が腐らないとか、頭がよくなるとか、四角錐にまつわるウワサは真実味にあふれています。

そこに突然登場した、このPp（ピューピー）なる未知数……。どこまで本当で、どこまでウソなのか、まったくわからないところにインチキの魅力がありそうです。

しかし、ここでは『BCDの数値×HL』だけが問題です。高速というからには、問（その74）の方法（足し算のループ）では不満なはず。どうにかして、HLレジスタの値をBCD化して、問（その75）のプログラムが使えるようにしなければなりません。

ここにあるプログラムは、HLレジスタの値をBCDに直し、問（その75）にあるプログラムの「BDATA」に入れるというものです。つまり、このプログラ

ム (HLBCD) をコールしてから問 (その75) のプログラムをコールすればいいわけです。

```
HLBIT :   DB     0,0,0            ←ビット別加算データ

HLBCD :   EX     DE,HL            ← DE＝変換したいデータ
          XOR    A
          LD     IX,BDATA+2       ←変換先
          LD     (IX+0),A
          LD     (IX-1),A
          LD     (IX-2),A
          LD     HL,HLBIT         ← (HLBIT)～(HLBIT+2) の初期化
          LD     (HL),A
          INC    HL
          LD     (HL),A
          INC    HL
          LD     (HL),1
          LD     B,16             ← B＝変換ビット数 (16ビット)
HLBLP :   LD     HL,HLBIT+2
          SRL    D
          RR     E
          CALL   C,BTADD          ←ビット=1のところだけビット別加算データを加算
          LD     A,(HL)           ←ビット別加算データを2倍する
          ADD    A,A
          DAA
          LD     (HL),A
          DEC    HL
          LD     A,(HL)
          ADC    A,A
          DAA
          LD     (HL),A
          DEC    HL
          LD     A,(HL)
          ADC    A,A
          DAA
          LD     (HL),A
          DJNZ   HLBLP
          RET

BTADD :   LD     A,(HL)           ←ビット別加算データを変換先に加算
          ADD    A,(IX+0)
          DAA
```

```
        LD      (IX+0),A
        DEC     HL
        LD      A,(HL)
        ADC     A,(IX-1)
        DAA
        LD      (IX-1),A
        DEC     HL
        LD      A,(HL)
        ADC     A,(IX-2)
        DAA
        LD      (IX-2),A
        LD      HL,HLBIT+2
        RET
```

プログラムの考え方は、HL レジスタを各ビット別に BCD 化して、16ビット分加算していくものです。ビット＝1の場合に加算する値（BCD 値）は次のようになります。

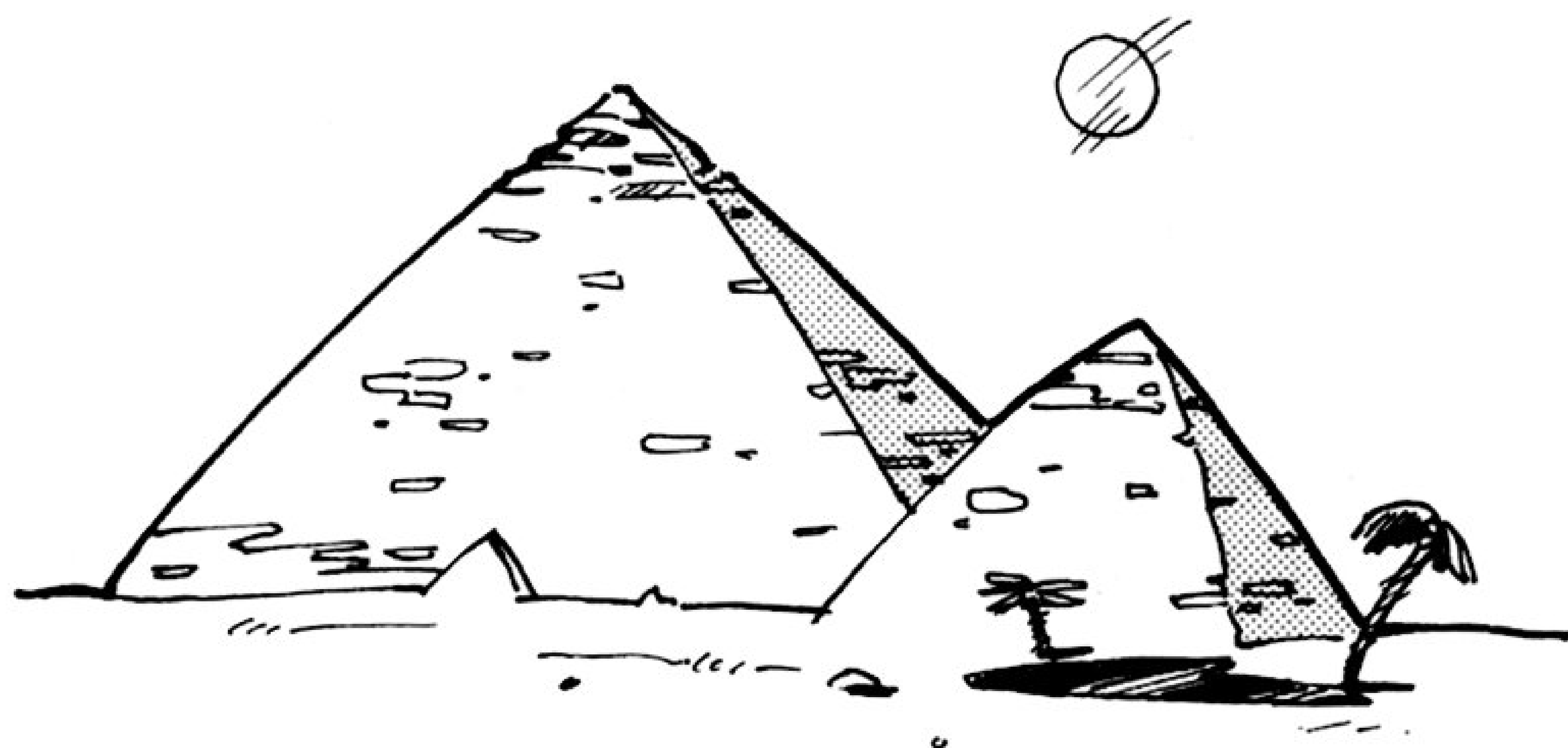
ビット 0=00,00,01	ビット 8=00,02,56
ビット 1=00,00,02	ビット 9=00,05,12
ビット 2=00,00,04	ビット 10=00,10,24
ビット 3=00,00,08	ビット 11=00,20,48
ビット 4=00,00,16	ビット 12=00,40,96
ビット 5=00,00,32	ビット 13=00,81,92
ビット 6=00,00,64	ビット 14=01,63,84
ビット 7=00,01,28	ビット 15=03,27,68

今回は、これらのビット別データを計算によって求めていますが、メモリにデータとして用意しておくと、「ビット別加算データを2倍する」代わりにそのデータアドレスを変更するだけで済むので、メモリ効率は悪くなりますがスピードはアップします。

また、問(その41)のような考え方で、HL レジスタの値を 10000、1000、100、10で減算するように割り、その商を BCD 化していくという方法もあります。ど

ちらの方法が処理が速いかは、その時のHLレジスタの値によって変わってきますので、状況に応じながら使い分けてください。

これでPp（ピューピー）が発見できるのであれば、盆と正月と誕生日とクリスマスが一度にやって来るでしょう。ついでに、宝くじにも当たるかもしれません……ネ!?



問 その77

BCDどうしの割り算（割る数の桁数固定）

鉄人88号がいいだろうか、それとも鉄腕ゼットがいいかな、やっぱりエイトマシンにしようか……。

ネーミングが決まらないから、なかなか設計にかかれない。人間が月へ行く時代だというのに、いまだに本物のロボットが作れないというのは、はなはだ遺憾である。自動車メーカーには、部分的にロボットらしきものがあるが、いつまで待ってもあれは歩かないようだ。

マ、順序としては鉄人88号から完成させるのが筋だろうな。もちろん、IC は Z80を使うことになる。では、さっそく設計にかかろう。

……ウ～ム、電波を受けてから手を動かすためには、高速な割り算をしなければならないのか。一応、割る数は 8888 に固定しよう。割られる数は不定だが、8桁だな。ということは、BCD による筆算タイプの割り算でないと無理だ。

しかし、減算ループによる割り算ならできそうだが、筆算のようにするには難しそうだぞ。設計の前に教えてもらわなければなるまい。

鉄人のために、よろしく頼む。

敷島博士（山梨）

答

鉄人88号……。名前はカッコイイけど、完成した鉄人が活躍する場所を捜すのに苦労しそうです。

まず、相手がいない。今どき、わざわざ目立つロボットを作って悪いことをしようなどという悪人はいないでしょう。おまけに、下手に歩けば道路はこわすし電線にも引っかかる。それではとロケットが火を噴けば、周りは火事になって大惨事……。

これでは、せっかくの夢がこわれそうです。余計なことを考えずに、BCD の筆算タイプの割り算を実現することにしましょう。

かけ算の時もそうでしたが、BCD では割り算を問（その68）のように二進数的な考え方で実行することはできません。そのため、商の桁ごとに減算のループによる割り算を行うことになります。

今回は、割られる数が8桁、割る数が 8888 に固定されているので、4バイトの BCD 数値を2バイト（4桁）の BCD 数値で割るという計算ですが、問（その68）に示されているように割られる数、割る数の桁数は割り算の重要なポイ

ントです。

商の桁数＝８－４＋１

また、BCDの計算では、その桁の商がメモリの上位４ビットに入るか、下位４ビットに入るかも区別しなければなりません。今回の場合は、商が５桁ですからメモリの下位４ビットから入れることになります。

ここではサンプルデータとして割られる数＝10967792としていますが、もし割り切れない数となった場合は小数点以下切り捨てです。ただし、小数点はユーザー側の区切りですから、ダミーのメモリ（中身＝０）を追加すれば、小数点以下何位まででも求めることができます。その場合、割る数（BDATA）や商（KEKKA）のメモリも計算に合わせて追加しなければなりません。

割り算の最後は常に切り捨てですから、四捨五入したい場合は問(その73)の方法で自由にしてください。

```
KEKKA :   DB      0,0,0                ←商が入る
ADATA :   DB      10H,96H,77H,92H      ←割られる数（８桁）
BDATA :   DB      88H,88H,0,0          ←割る数（上位４桁）
                                         下４桁は桁合わせのためのダミー

WARIX :   LD      IX,KEKKA
          LD      C,0                  ←C＝桁ごとの商
          CALL    WARI1                ←下位４ビットの商を求める
          CALL    WARI2                ←上位４ビット／下位４ビットの商を求める
          CALL    WARI2                ←           //
          RET

WARI2 :   LD      C,0
          CALL    WAXCC
          SLA     C                    ←商を上位４ビットに移す
          SLA     C
          SLA     C
          SLA     C
WARI1 :   CALL    WAXCC
          LD      (IX+0),C
          INC     IX
          RET
```

```
WAXCC :   LD      DE,ADATA+3    ① ←桁ごとの商を求める割り算
          LD      HL,BDATA+3    ②
          OR      A
WACL0 :   LD      B,4           ③ ←BはBCD数値のバイト数
          PUSH    DE               減算ループによる割り算
          PUSH    HL
SBCL0 :   LD      A,(DE)
          SBC     A,(HL)
          DAA
          LD      (DE),A
          DEC     DE
          DEC     HL
          DJNZ    SBCL0
          POP     HL
          POP     DE
          JR      C,WAED0
          INC     C
          JR      WACL0
WAED0 :   LD      B,4           ③ ←減算し過ぎた分を加算する
          OR      A
ADCL0 :   LD      A,(DE)
          ADC     A,(HL)
          DAA
          LD      (DE),A
          DEC     DE
          DEC     HL
          DJNZ    ADCL0
          XOR     A
          LD      B,4           ③ ←割る数を次の桁用に右シフトする
HLRRD :   INC     HL
          RRD
          DJNZ    HLRRD
          RET
```

なお、プログラムを実行すると、割られる数(ADATA)は割り算の余りとなり、割る数(BDATA)は破壊されます。必要があれば、退避してから実行してください。

そういえば、敷島博士の写真が同封されていましたが、年老いて、博士というよりまるで仙人のような風貌でした……。

問 その78

BCDどうしの割り算(割る数の桁数不定)

わらわは卑弥呼……。その昔、わらわはこの耶馬台国を治めておったのじゃが、いまだにわらわが治めた場所がわからぬというではないか。

北九州とか佐賀とか大和地方とか、説は色々と出ているようだが、実はわらわも古いことなのでよく覚えていないのじゃ。というより、わらわの記憶にはインプットされなかったのじゃ。

おっと、わらわは本物の卑弥呼でもなければ、その生まれ変わりでもない。実を申せば、わらわは様々な資料を基に作られたコンピュータ卑弥呼にあるぞ。スイッチポンでわらわは見事に蘇り、そなたに話しかけるのじゃ。

わらわを作ったのは、是戸八丸（ぜとはちまる）というヒマ人じゃ。わらわが彼と話したところによると、耶馬台国の距離と方角と面積のデータは入れてあるのだそうじゃ。だから、桁数不定の割り算ができれば、その地域が特定できるということじゃ。

データはBCDによる数値で、桁数は最大で12桁じゃ。もしも、そなたにこの耶馬台国の秘密が解き明かせるなら、わらわが人間に生まれ変わった時、そなたの嫁になってやってもよいぞよ。

卑弥呼の部屋（奈良）

答

嫁……だなんて、あまりに恐れ多くて、丁重にご辞退申し上げます。凡人には凡人の歩む道があり、BCDによる桁数不定の割り算ごときで嫁に来られたのでは身体がいくつあっても足りませぬゆえ。

とはいえ、汎用のプログラムとして、桁数不定の割り算程度は作っておいたほうが便利でしょう。もちろん、まったくの不定ではプログラムになりませんが、とりあえず最大桁数＝12という条件があるので、これに準じてプログラムを組むことにします。

割り算自体の考え方は問（その77）と共通です。しかし、用意するメモリ数が違うため、問（その77）のプログラムの「WARI2」以下の部分を次のように変更して使用します。

```
① → LD     DE,ADATA+5
② → LD     HL,BDATA+5
③ → LD     B,6
```

したがって、ここにあるプログラムは割り算ルーチンに入る前のデータ初期化と、変更した「WARI2」以下を利用して割り算をするプログラムです。

```
KEKKA :   DB     0,0,0,0,0,0                  ←商が入る
ADATA :   DB     15H,18H,51H,83H,94H,00H      ←割られる数（例）
BDATA :   DB     0,0,0,0,12H,30H              ←割る数（例）

WARIX :   XOR    A
          LD     HL,KEKKA                     ←商をクリアする
          LD     B,6
WAIL0 :   LD     (HL),A
          INC    HL
          DJNZ   WAIL0
          LD     E,1                          ←E＝商の桁数（計算回数）
          LD     C,12                         ←C＝割る数の桁数
WAIL1 :   LD     A,(BDATA)                    ←割る数のトップ4ビットがゼロで
          AND    0F0H                           なくなるまで桁シフトを行う。同
          JR     NZ,WAIE1                       時にEレジスタに商の桁数を求め
          INC    E                              ている
          LD     HL,BDATA+5
          LD     B,6
WAIL2 :   RLD
          DEC    HL
          DJNZ   WAIL2
          DEC    C
          JR     NZ,WAIL1
          RET                                 ←12桁ともゼロならリターン

WAIE1 :   LD     IX,KEKKA+5                   ←商の入るアドレスを決める
          SRL    E                            ←商の桁数÷2が商に必要なメモリ
          LD     B,E                            数
          JR     Z,WAIE2
WAIL3 :   DEC    IX
          DJNZ   WAIL3
          LD     B,E
WAIE2 :   LD     C,0                          ←桁ごとの商の初期値
          PUSH   BC
          CALL   C,WARI1                      ←商の桁数が奇数の時にコール
          POP    BC
          LD     A,B
```

```
            OR      A
            RET     Z               ←商の桁数＝1の時はリターン
WAXLP :     PUSH    BC
            CALL    WARI2           ←商2桁を求める
            POP     BC
            DJNZ    WAXLP
            RET
```

割る数の桁を求め商の桁数を決定するという考え方は、問(その68) と同じです。また、筆算でも桁数が多ければ桁数を数えて商の先頭位置を決めるように、各桁の商が入るメモリの位置を最初に確定しなければなりません。プログラムでいうと「WAIE1」の部分ですが、レジスタの内容と役目をよく捉えるようにしてください。

これで桁数不定の割り算が可能になったわけですが、例のコンピュータ卑弥呼に利用した後の結果については一切責任は取れませんので悪しからず……。



問 その79　BCDに関するミニ・テクニック

人間はコンピュータを作った。そして、それは人工知能へと発展しつつある。最近ではファジー理論という、人間らしい曖昧さを求めた研究も盛んに行われているというではないか。つまり、人間は自分と同じことをコンピュータにさせることで、生命を創造した神になろうとしているのだ。

……だが、おいらの存在を忘れてもらっちゃ困るぜ。コホン!!　おいらはコンピュータなどには絶対にマネのできない第三の知能さ。おいらの正体はまだ発見されていないようだが、誰でもおいらの存在は知っているんだ。

おいらの名は『夢』……。夢を司る脳に住んでいるのさ。いくらコンピュータが人間のマネをしたところで、夢は見れまい。そこに神の偉大さがあるのだ。

だけど、おいらにも夢がある。それはおいらの存在をコンピュータ化してもらうことなんだ。そのためには、BCD どうしの比較やゼロチェックが自由自在に、しかもスバヤクできなければならない。

人工夢脳のために、こういった BCD のミニ・テクを公開する気はないだろうか。もっとも、それは人工夢脳へのスタートに過ぎないが……。

夢見る夢（夢脳）

答

近くて遠い夢の国……。
行けそうで行けない夢の国……。
現実のようでどこかが違う夢の国……。

夢をプログラムで実現できるかどうかはわかりませんが、その前に夢を録画するビデオを開発してほしいものです。見ている時は真実そのもの、しかし実際にはデタラメで支離滅裂でウソと幻に包まれたナゾの世界……。そのナゾの解明に役立てるなら、BCD のミニ・テクなどいくらでも公開しましょう。

(1)　BCD どうしの比較

基本的には減算をすればいいのですが、下位桁から減算をしていくと全メモリを減算しないと比較できません。人間的な思考法では、上位桁から比較をするほうが自然です。このプログラムは「CP　DE(BCD のアドレス)，HL(BCD のアドレス)」を実行し、ゼロ／キャリーフラグで判定を示すものです。

例えば「CP　①,②　」をしたければ、DE レジスタ=ADATA、HL レジスタ=BDATA として CPBCD をコールします。

```
ADATA :   DB      12H,34H,56H,78H,90H,12H   ←①
BDATA :   BD      11H,22H,33H,44H,55H,66H   ←②

CPBCD :   LD      B,6                       ←B=BCD データのバイト数
CPBLP :   LD      A,(DE)
          CP      (HL)
          RET     NZ
          INC     HL
          INC     DE
          DJNZ    CPBLP
          RET
```

(2)　BCD のゼロチェック

すべての桁（メモリ）がゼロかどうかを調べるわけですが、これも上位から調べるほうが人間的でしょう。HL レジスタに調べたい BCD のアドレスを入れてコールしてください。

```
BCDZF :   XOR     A
          LD      B,6      ←BCD データバイト数
BCZLP :   OR      (HL)
          RET     NZ
          INC     HL
          DJNZ    BCZLP
          RET
```

(3)　BCD の符号について

BCD の数の中には符号を含むことができませんから、符号が必要な場合には符号用のメモリを1バイト用意しなければなりません。例えば、0 ならプラス、1 ならマイナスと決めておいて、加減算の場合にはその符号と照らし合わせてから実行するようにするのです。

ただし、計算を加減算だけに限定すれば、符号付き数値のように扱うこと

もできます。例えば、2桁のBCD数値で15とあれば、85を −15と考えてもいいわけです。

(例)

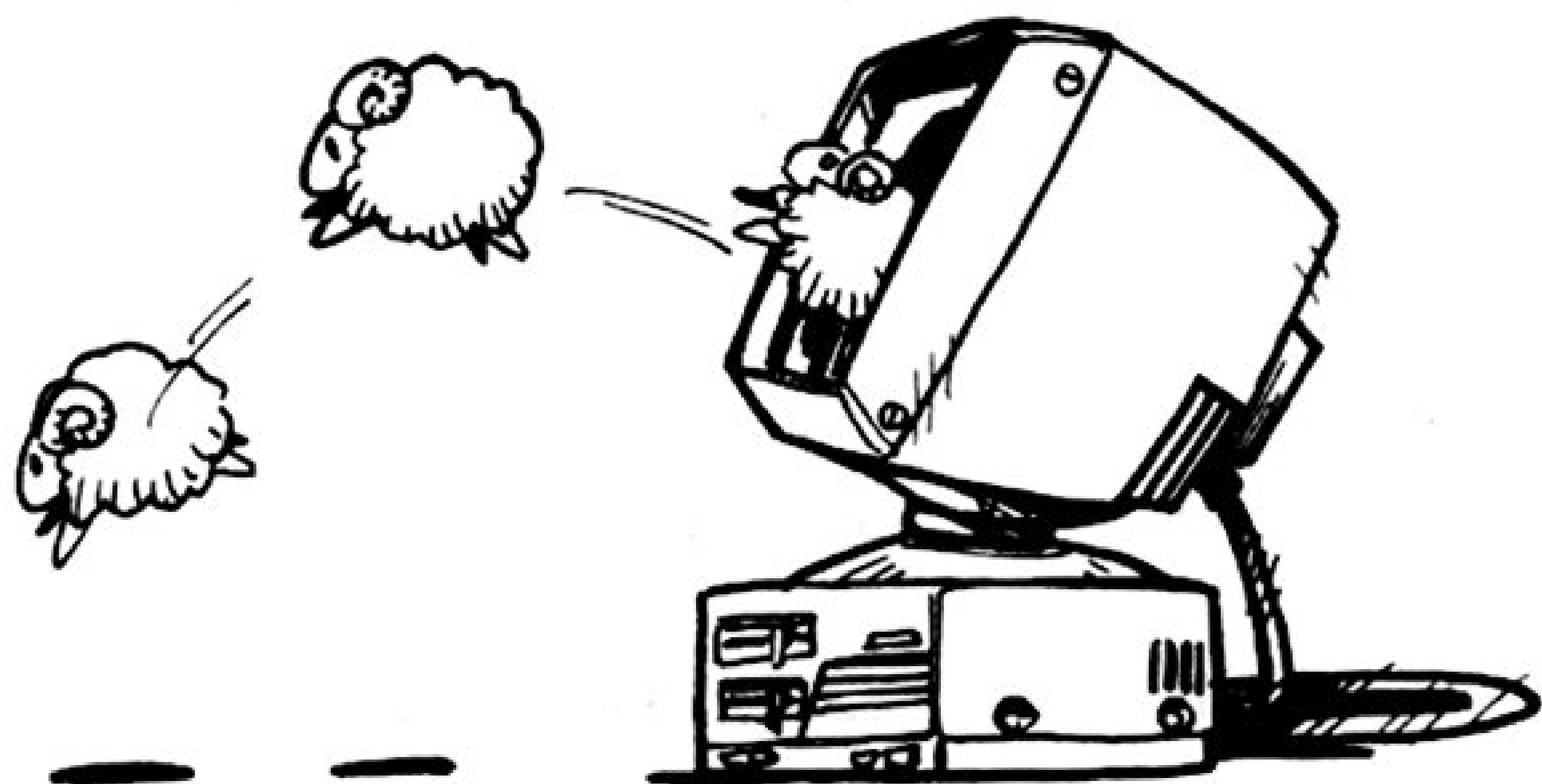
20−15=05

20+85=05 ←100以上は計算されない

このような符号変換は、いわばBCD版NEG命令ということになります。次のプログラムは3バイト用のBCD版NEG命令です。HLレジスタにBCD数値の最下位アドレスを入れてコールしてください。

```
BCNEG :  XOR    A
         LD     B,3          ← BCDデータのバイト数
BNGLP :  SBC    A,(HL)
         DAA
         LD     (HL),A
         DEC    HL
         LD     A,0
         DJNZ   BNGLP
         RET
```

いずれも大したテクニックではありませんが、BCD数値を自在に扱うための基礎として覚えておくと便利です。その程度ですから、これが夢の人工夢脳の開発に役に立つことは夢にも思えません……。



問 その80 テーブル処理で複雑な計算を

放浪……。そこには未知のものに出会える夢がある。しかし、最近はテレビが発達し過ぎたせいで、夢が減ってしまった。

昔は『知床旅情』なんていう歌は、北海道へ旅した者が覚えてくる歌だったのだ。だから、この歌がテレビを通じて流行ったとたん、夢が1つ消えた。

かつて、私は日本最北端の島、礼文島を旅した。ユースホステルでは、元フォークグループの若者が歌を教えていた。それは、『旅の終わり』、『島を愛す』という2つの歌だった。これらは、少なくともメジャーな歌にはならなかった。

もしかすると、この歌はもう誰も知らないかもしれない……。私は、ときどき口ずさんでは、たった一人の優越感に浸っている。

私は、さすらいのプログラマー……。もちろん、かけ算は知っている。実は X^2を多用するプログラムがあるのだが、毎回計算しているので時間がかかる。きっと、もっといい方法があると思うのだが、現状ではわからない。

しかし、マシン語には未知のテクニックを発見できる夢がある。まるで放浪の旅をしているようだ。私は、そんなマシン語が好きだ。

さすらい人（礼文島）

答

テレビというのは、夢を与えてくれているようで、実は夢を奪っているのです。なぜなら、夢は想像の中から生まれるものだから……。

マシン語の世界は限りない想像の世界です。そこには、高級言語では味わうことのできない未知の魅力と無限の可能性があります。もしかすると、最も身近な未踏の秘境かもしれません。

……が、マシン語で秘境探検をするには発想の転換が必要です。この X^2も、多用するからには、かけ算処理を省きたいものです。次のプログラムは、Aレジスタ（0～255）を二乗し、その結果をHLレジスタに求めるというものです。

```
KDATA :   DW      0,1,4,9,16,25              ←0～255を二乗した数値をデー
          DW      36,49,64,81,100              タとして用意しておく
                   :    :
          DW      63504,64009,64516,65025

XTIMX :   LD      L,A                        ←A²の結果をHLに求める
          LD      H,0
```

```
        ADD     HL,HL
        LD      DE,KDATA
        ADD     HL,DE
        LD      E,(HL)
        INC     HL
        LD      D,(HL)
        EX      DE,HL
        RET
```

プログラム自体は、問(その24) でジャンプ・テーブルとして用いたものと似ています。しかし、アイデアは似ていても内容はまったく違います。さらに、このX²という計算も見本の１つに過ぎません。この用法の本当の価値は、複雑な計算をさせた場合に現れてくるのです。

また、計算結果を直接 BCD にすることもできます。つまり、テーブルの内容に工夫を凝らせば、いくらでも応用の範囲は広がるということです。人跡未踏のテクニックというわけではありませんが、マシン語のちょっとした秘境といえるでしょう。

しかし、もとの値が BCD の場合は、簡単にテーブルを利用できません。テーブルは十六進数のアドレスで示されますから、BCD のままでは虫喰いテーブル (A〜F のあるアドレスを使用しないテーブル) となってしまうからです。そこで、参考までに 0〜65535 の BCD 数値を十六進数に変換し、HL レジスタに入れるプログラムを載せておきます。

```
BCDDT :  DB      06H,55H,35H      ← BCD の数値 (例)

BCDHL :  LD      IX,BCDDT
         LD      HL,0
         LD      A,(IX+0)
         LD      DE,10000
         CALL    KAI4B            ←10000の位の計算
         LD      A,(IX+1)
         LD      DE,1000
         CALL    JOI4B            ←1000の位の計算
         LD      DE,100           ←100の位の計算
         CALL    KAI4B
```

```
            LD      A,(IX+2)
            LD      DE,10
            CALL    JOI4B          ←10の位の計算
            AND     00001111B
            LD      E,A
            ADD     HL,DE          ←1の位の計算
            RET

JOI4B :     LD      C,A            ←上位4ビットの計算
            RRCA
            RRCA
            RRCA
            RRCA
            CALL    KAI4B
            LD      A,C
            RET
KAI4B :     AND     00001111B      ←下位4ビットの計算
            LD      B,A
AHDLP :     ADD     HL,DE
            DJNZ    AHDLP
            RET
```

このプログラムを利用すれば、例えば「0〜99 の BCD 数値」→「計算結果＝6バイト（12桁）の BCD 数値」などといったテーブルも簡単に実現できます。もちろん、テーブルのために使用するだけでなく、単なる変換ルーチンとして BCD 数値のままでは不便という場合に活用しても構いません。

テーブルというのは応用範囲が大変広いテクニックですから、プログラムが複雑になったりダラダラと長くなりそうなときは、テーブル化を検討する価値があります。もしかすると、まったく新しい用法や秘術に出会えるかもしれません。

『新テクの陰にテーブルあり』

これも、マシン語の極意の1つといえるでしょう。マシン語にはこのようにメジャーなテクニックもあれば、日陰の鈴蘭のようなマイナーなテクニックもあります。私も、そんなマシン語が好きです……。

三の章　正しいマシン語のために

次ページ以降に掲載するマシン語ニモニック一覧表は、日本電気株式会社の「μCOM-82インストラクション活用表」より転載いたしました。

〔注1〕 ニモニック中の略号の意味

略号	意味
d, n	8ビット・イミーディエート・データ
e	相対アドレシングの変位置
e−2	eの実効変位置

〔注2〕 フラグの記号の意味

記号		意味
•		影響受けない
0		リセット
1		セット
×		不定
↕		演算結果に従った影響受ける(①②③あり＝下に説明)
P		1＝偶数パリティ，0＝奇数パリティ
V		1＝オーバーフローあり，0＝オーバーフローなし
IFF		P/V フラグ ← (IFF)
↕	①	BC−1＝0 ならば P/V＝0，その他 P/V＝1
↕	②	A＝(HL) ならば Z＝1，その他 Z＝0
↕	③	B−1＝0 ならば Z＝1，その他 Z＝0

〔注3〕 OPコードの注記の意味

Ⓐ		Ⓑ		Ⓒ		Ⓓ		Ⓔ		Ⓕ		Ⓖ				Ⓗ	
Reg	ss dd	Reg	qq	Reg	pp	Reg	rr	Reg	r,r′	Bit	b	cc		Condition	Flag	P	t
BC	00	BC	00	BC	00	BC	00	B	000	0	000	000	NZ	Non Zero	Z	00H	000
DE	01	DE	01	DE	01	DE	01	C	001	1	001	001	Z	Zero	Z	08H	001
HL	10	HL	10	IX	10	IY	10	D	010	2	010	010	NC	Non Carry	C	10H	010
SP	11	AF	11	SP	11	SP	11	E	011	3	011	011	C	Carry	C	18H	011
								H	100	4	100	100	PO	Parity Odd	P/V	20H	100
								L	101	5	101	101	PE	Parity Even	P/V	28H	101
								A	111	6	110	110	P	Sign Positive	S	30H	110
										7	111	111	M	Sign Negative	S	38H	111

d, n：8ビット・イミーディエート・データ
e　：相対アドレシングの変位置
e−2：eの実効変位置

Z80 ニモニック表

命令群	ニモニック	オペレーション	フラグ S	Z	H	P/V	N	C	バイト	ステート	OPコード 76	543	210	
8ビット・ロード命令	LD r, r'	r←r'	•	•	•	•	•	•	1	4	01	r	r'	Ⓔ
	LD r, n	r←n	•	•	•	•	•	•	2	7	00	r	110	Ⓔ
											←	n	→	
	LD r, (HL)	r←(HL)	•	•	•	•	•	•	1	7	01	r	110	Ⓔ
	LD r, (IX+d)	r←(IX+d)	•	•	•	•	•	•	3	19	11	011	101	
											01	r	110	Ⓔ
											←	d	→	
	LD r, (IY+d)	r←(IY+d)	•	•	•	•	•	•	3	19	11	111	101	
											01	r	110	Ⓔ
											←	d	→	
	LD (HL), r	(HL)←r	•	•	•	•	•	•	1	7	01	110	r	Ⓔ
	LD (IX+d), r	(IX+d)←r	•	•	•	•	•	•	3	19	11	011	101	
											01	110	r	Ⓔ
											←	d	→	
	LD (IY+d), r	(IY+d)←r	•	•	•	•	•	•	3	19	11	111	101	
											01	110	r	Ⓔ
											←	d	→	
	LD (HL), n	(HL)←n	•	•	•	•	•	•	2	10	00	110	110	
											←	n	→	
	LD (IX+d), n	(IX+d)←n	•	•	•	•	•	•	4	19	11	011	101	
											00	110	110	
											←	d	→	
											←	n	→	
	LD (IY+d), n	(IY+d)←n	•	•	•	•	•	•	4	19	11	111	101	
											00	110	110	
											←	d	→	
											←	n	→	
	LD A,(BC)	A←(BC)	•	•	•	•	•	•	1	7	00	001	010	
	LD A,(DE)	A←(DE)	•	•	•	•	•	•	1	7	00	011	010	
	LD A,(nn)	A←(nn)	•	•	•	•	•	•	3	13	00	111	010	
											←	n	→	
											←	n	→	
	LD (BC), A	(BC)←A	•	•	•	•	•	•	1	7	00	000	010	
	LD (DE), A	(DE)←A	•	•	•	•	•	•	1	7	00	010	010	
	LD (nn), A	(nn)←A	•	•	•	•	•	•	3	13	00	110	010	
											←	n	→	
											←	n	→	
	LD A, I	A←I	↕	↕	0	IFF	0	•	2	9	11	101	101	
											01	010	111	
	LD A, R	A←R	↕	↕	0	IFF	0	•	2	9	11	101	101	
											01	011	111	
	LD I, A	I←A	•	•	•	•	•	•	2	9	11	101	101	
											01	000	111	
	LD R, A	R←A	•	•	•	•	•	•	2	9	11	101	101	
											01	001	111	
16ビット・ロード命令	LD dd, nn	dd←nn	•	•	•	•	•	•	3	10	00	dd0	001	Ⓐ
											←	n	→	
											←	n	→	
	LD IX, nn	IX←nn	•	•	•	•	•	•	4	14	11	011	101	
											00	100	001	
											←	n	→	
											←	n	→	

命令群	ニモニック	オペレーション	フラグ S	Z	H	P/V	N	C	バイト	ステート	OPコード 76	543	210	
16ビット・ロード命令	LD IY, nn	IY←nn	•	•	•	•	•	•	4	14	11 00 ← ←	111 100 n n	101 001 → →	
	LD HL, (nn)	H←(nn+1) L←(nn)	•	•	•	•	•	•	3	16	00 ← ←	101 n n	010 → →	
	LD dd, (nn)	dd_H←(nn+1) dd_L←(nn)	•	•	•	•	•	•	4	20	11 01 ← ←	101 dd1 n n	101 011 → →	 Ⓐ
	LD IX, (nn)	IX_H←(nn+1) IX_L←(nn)	•	•	•	•	•	•	4	20	11 00 ← ←	011 101 n n	101 010 → →	
	LD IY, (nn)	IY_H←(nn+1) IY_L←(nn)	•	•	•	•	•	•	4	20	11 00 ← ←	111 101 n n	101 010 → →	
	LD (nn), HL	(nn+1)←H (nn)←L	•	•	•	•	•	•	3	16	00 ← ←	100 n n	010 → →	
	LD (nn), dd	(nn+1)←dd_H (nn)←dd_L	•	•	•	•	•	•	4	20	11 01 ← ←	101 dd0 n n	101 011 → →	 Ⓐ
	LD (nn), IX	(nn+1)←IX_H (nn)←IX_L	•	•	•	•	•	•	4	20	11 00 ← ←	011 100 n n	101 010 → →	
	LD (nn), IY	(nn+1)←IY_H (nn)←IY_L	•	•	•	•	•	•	4	20	11 00 ← ←	111 100 n n	101 010 → →	
	LD SP, HL	SP←HL	•	•	•	•	•	•	1	6	11	111	001	
	LD SP, IX	SP←IX	•	•	•	•	•	•	2	10	11 11	011 111	101 001	
	LD SP, IY	SP←IY	•	•	•	•	•	•	2	10	11 11	111 111	101 001	
	PUSH qq	(SP−2)←qq_L (SP−1)←qq_H	•	•	•	•	•	•	1	11	11	qq0	101	Ⓑ
	PUSH IX	(SP−2)←IX_L (SP−1)←IX_H	•	•	•	•	•	•	2	15	11 11	011 100	101 101	
	PUSH IY	(SP−2)←IY_L (SP−1)←IY_H	•	•	•	•	•	•	2	15	11 11	111 100	101 101	
	POP qq	qq_H←(SP+1) qq_L←(SP)	•	•	•	•	•	•	1	10	11	qq0	001	Ⓑ
	POP IX	IX_H←(SP+1) IX_L←(SP)	•	•	•	•	•	•	2	14	11 11	011 100	101 001	
	POP IY	IY_H←(SP+1) IY_L←(SP)	•	•	•	•	•	•	2	14	11 11	111 100	101 001	
エクスチェンジ命令	EX DE, HL	DE↔HL	•	•	•	•	•	•	1	4	11	101	011	
	EX AF, AF'	AF↔AF'	•	•	•	•	•	•	1	4	00	001	000	

命令群	ニモニック	オペレーション	フラグ						バイト	ステート	OPコード	
			S	Z	H	P/V	N	C			76 543 210	
エクスチェンジ命令	EXX	(BC↔BC' DE↔DE' HL↔HL')	•	•	•	•	•	•	1	4	11 011 001	
	EX (SP), HL	H↔(SP+1) L↔(SP)	•	•	•	•	•	•	1	19	11 100 011	
	EX (SP), IX	IX_H↔(SP+1) IX_L↔(SP)	•	•	•	•	•	•	2	23	11 011 101 11 100 011	
	EX (SP), IY	IY_H↔(SP+1) IY_L↔(SP)	•	•	•	•	•	•	2	23	11 111 101 11 100 011	
ブロック転送命令	LDI	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1	•	•	0	① ↕	0	•	2	16	11 101 101 10 100 000	
	LDIR	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1 until BC=0	•	•	0	0	0	•	2	21 if BC≠0 16 if BC=0	11 101 101 10 110 000	
	LDD	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1	•	•	0	① ↕	0	•	2	16	11 101 101 10 101 000	
	LDDR	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1 until BC=0	•	•	0	0	0	•	2	21 if BC≠0 16 if BC=0	11 101 101 10 111 000	
ブロック・サーチ命令	CPI	A−(HL) HL←HL+1 BC←BC−1	↕	② ↕	↕	① ↕	1	•	2	16	11 101 101 10 100 001	
	CPIR	A−(HL) HL←HL+1 BC←BC−1 until A=(HL) or BC=0	↕	② ↕	↕	① ↕	1	•	2	21 if BC≠0 and A≠(HL) 16 if BC=0 or A=(HL)	11 101 101 10 110 001	
	CPD	A−(HL) HL←HL−1 BC←BC−1	↕	② ↕	↕	① ↕	1	•	2	16	11 101 101 10 101 001	
	CPDR	A−(HL) HL←HL−1 BC←BC−1 until A=(HL) or BC=0	↕	② ↕	↕	① ↕	1	•	2	21 if BC≠0 and A≠(HL) 16 if BC=0 or A=(HL)	11 101 101 10 111 001	
8ビット算術論理演算命令	ADD A, r	A←A+r	↕	↕	↕	V	0	↕	1	4	10 000 r	Ⓔ
	ADD A, n	A←A+n	↕	↕	↕	V	0	↕	2	7	11 000 110 ← n →	
	ADD A, (HL)	A←A+(HL)	↕	↕	↕	V	0	↕	1	7	10 000 110	
	ADD A, (IX+d)	A←A+(IX+d)	↕	↕	↕	V	0	↕	3	19	11 011 101 10 000 110 ← d →	

命令群	ニモニック	オペレーション	フラグ						バイト	ステート	OPコード			
			S	Z	H	P/V	N	C			76	543	210	
	ADD, A, (IY+d)	A←A+(IY+d)	↕	↕	↕	V	0	↕	3	19	11	111	101	
											10	000	110	
											←	d	→	
	ADC A, r	A←A+r+CY	↕	↕	↕	V	0	↕	1	4	10	001	r	Ⓔ
	ADC A, n	A←A+n+CY	↕	↕	↕	V	0	↕	2	7	11	001	110	
											←	n	→	
	ADC A, (HL)	A←A+(HL)+CY	↕	↕	↕	V	0	↕	1	7	10	001	110	
	ADC A, (IX+d)	A←A+(IX+d)+CY	↕	↕	↕	V	0	↕	3	19	11	011	101	
											10	001	110	
											←	d	→	
	ADC A, (IY+d)	A←A+(IY+d)+CY	↕	↕	↕	V	0	↕	3	19	11	111	101	
											10	001	110	
											←	d	→	
	SUB r	A←A−r	↕	↕	↕	V	1	↕	1	4	10	010	r	Ⓔ
	SUB n	A←A−n	↕	↕	↕	V	1	↕	2	7	11	010	110	
											←	n	→	
8ビット算術論理演算命令	SUB (HL)	A←A−(HL)	↕	↕	↕	V	1	↕	1	7	10	010	110	
	SUB (IX+d)	A←A−(IX+d)	↕	↕	↕	V	1	↕	3	19	11	011	101	
											10	010	110	
											←	d	→	
	SUB (IY+d)	A←A−(IY+d)	↕	↕	↕	V	1	↕	3	19	11	111	101	
											10	010	110	
											←	d	→	
	SBC A, r	A←A−r−CY	↕	↕	↕	V	1	↕	1	4	10	011	r	Ⓔ
	SBC A, n	A←A−n−CY	↕	↕	↕	V	1	↕	2	7	11	011	110	
											←	n	→	
	SBC A, (HL)	A←A−(HL)−CY	↕	↕	↕	V	1	↕	1	7	10	011	110	
	SBC A, (IX+d)	A←A−(IX+d)−CY	↕	↕	↕	V	1	↕	3	19	11	011	101	
											10	011	110	
											←	d	→	
	SBC A, (IY+d)	A←A−(IY+d)−CY	↕	↕	↕	V	1	↕	3	19	11	111	101	
											10	011	110	
											←	d	→	
	AND r	A←A∧r	↕	↕	1	P	0	0	1	4	10	100	r	Ⓔ
	AND n	A←A∧n	↕	↕	1	P	0	0	2	7	11	100	110	
											←	n	→	
	AND (HL)	A←A∧(HL)	↕	↕	1	P	0	0	1	7	10	100	110	
	AND (IX+d)	A←A∧(IX+d)	↕	↕	1	P	0	0	3	19	11	011	101	
											10	100	110	
											←	d	→	
	AND (IY+d)	A←A∧(IY+d)	↕	↕	1	P	0	0	3	19	11	111	101	
											10	100	110	
											←	d	→	
	OR r	A←A∨r	↕	↕	0	P	0	0	1	4	10	110	r	Ⓔ
	OR n	A←A∨n	↕	↕	0	P	0	0	2	7	11	110	110	
											←	n	→	
	OR (HL)	A←A∨(HL)	↕	↕	0	P	0	0	1	7	10	110	110	
	OR (IX+d)	A←A∨(IX+d)	↕	↕	0	P	0	0	3	19	11	011	101	
											10	110	110	
											←	d	→	
	OR (IY+d)	A←A∨(IY+d)	↕	↕	0	P	0	0	3	19	11	111	101	
											10	110	110	
											←	d	→	
	XOR r	A←A⊻r	↕	↕	0	P	0	0	1	4	10	101	r	Ⓔ

命令群	ニモニック	オペレーション	フラグ						バイト	ステート	OPコード			
			S	Z	H	P/V	N	C			76	543	210	
8ビット算術論理演算命令	XOR n	A←A⊻n	↕	↕	0	P	0	0	2	7	11	101	110	
											←	n	→	
	XOR (HL)	A←A⊻(HL)	↕	↕	0	P	0	0	1	7	10	101	110	
	XOR (IX+d)	A←A⊻(IX+d)	↕	↕	0	P	0	0	3	19	11	011	101	
											10	101	110	
											←	d	→	
	XOR (IY+d)	A←A⊻(IY+d)	↕	↕	0	P	0	0	3	19	11	111	101	
											10	101	110	
											←	d	→	
	CP r	A−r	↕	↕	↕	V	1	↕	1	4	10	111	r	Ⓔ
	CP n	A−n	↕	↕	↕	V	1	↕	2	7	11	111	110	
											←	n	→	
	CP (HL)	A−(HL)	↕	↕	↕	V	1	↕	1	7	10	111	110	
	CP (IX+d)	A−(IX+d)	↕	↕	↕	V	1	↕	3	19	11	011	101	
											10	111	110	
											←	d	→	
	CP (IY+d)	A−(IY+d)	↕	↕	↕	V	1	↕	3	19	11	111	101	
											10	111	110	
											←	d	→	
	INC r	r←r+1	↕	↕	↕	V	0	•	1	4	00	r	100	Ⓔ
	INC (HL)	(HL)←(HL)+1	↕	↕	↕	V	0	•	1	11	00	110	100	
	INC (IX+d)	(IX+d)←(IX+d)+1	↕	↕	↕	V	0	•	3	23	11	011	101	
											00	110	100	
											←	d	→	
	INC (IY+d)	(IY+d)←(IY+d)+1	↕	↕	↕	V	0	•	3	23	11	111	101	
											00	110	100	
											←	d	→	
	DEC r	r←r−1	↕	↕	↕	V	1	•	1	4	00	r	101	Ⓔ
	DEC (HL)	(HL)←(HL)−1	↕	↕	↕	V	1	•	1	11	00	110	101	
	DEC (IX+d)	(IX+d)←(IX+d)−1	↕	↕	↕	V	1	•	3	23	11	011	101	
											00	110	101	
											←	d	→	
	DEC (IY+d)	(IY+d)←(IY+d)−1	↕	↕	↕	V	1	•	3	23	11	111	101	
											00	110	101	
											←	d	→	
16ビット算術演算命令	ADD HL, ss	HL←HL+ss	•	•	×	•	0	↕	1	11	00	ss1	001	Ⓐ
	ADC HL, ss	HL←HL+ss+CY	↕	↕	×	V	0	↕	2	15	11	101	101	
											01	ss1	010	Ⓐ
	SBC HL, ss	HL←HL−ss−CY	↕	↕	×	V	1	↕	2	15	11	101	101	
											01	ss0	010	Ⓐ
	ADD IX, pp	IX←IX+pp	•	•	×	•	0	↕	2	15	11	011	101	
											00	pp1	001	Ⓒ
	ADD IY, rr	IY←IY+rr	•	•	×	•	0	↕	2	15	11	111	101	
											00	rr1	001	Ⓓ
	INC ss	ss←ss+1	•	•	•	•	•	•	1	6	00	ss0	011	Ⓐ
	INC IX	IX←IX+1	•	•	•	•	•	•	2	10	11	011	101	
											00	100	011	
	INC IY	IY←IY+1	•	•	•	•	•	•	2	10	11	111	101	
											00	100	011	
	DEC ss	ss←ss−1	•	•	•	•	•	•	1	6	00	ss1	011	Ⓐ
	DEC IX	IX←IX−1	•	•	•	•	•	•	2	10	11	011	101	
											00	101	011	
	DEC IY	IY←IY−1	•	•	•	•	•	•	2	10	11	111	101	
											00	101	011	

命令群	ニモニック	オペレーション	S	Z	H	P/V	N	C	バイト	ステート	76	543	210	
アキュムレータ操作命令	DAA	Decimal Adjust Acc	↕	↕	↕	P	•	↕	1	4	00	100	111	
	CPL	A←$\overline{A}$	•	•	1	•	1	•	1	4	00	101	111	
	NEG	A←$\overline{A}$+1	↕	↕	↕	V	1	↕	2	8	11 01	101 000	101 100	
	CCF	CY←$\overline{CY}$	•	•	×	•	0	↕	1	4	00	111	111	
	SCF	CY←1	•	•	0	•	0	1	1	4	00	110	111	
CPUコントロール命令	NOP	No Operation	•	•	•	•	•	•	1	4	00	000	000	
	HALT	CPU Halted	•	•	•	•	•	•	1	4	01	110	110	
	DI	IFF←0	•	•	•	•	•	•	1	4	11	110	011	
	EI	IFF←1	•	•	•	•	•	•	1	4	11	111	011	
	IM 0	Set Interrupt Mode 0	•	•	•	•	•	•	2	8	11 01	101 000	101 110	
	IM 1	Set Interrupt Mode 1	•	•	•	•	•	•	2	8	11 01	101 010	101 110	
	IM 2	Set Interrupt Mode 2	•	•	•	•	•	•	2	8	11 01	101 011	101 110	
ローテート・シフト命令	RLCA	CY ← [7 ← 0] A	•	•	0	•	0	↕	1	4	00	000	111	
	RLA	CY ← [7 ← 0] A	•	•	0	•	0	↕	1	4	00	010	111	
	RRCA	[7 → 0] → CY	•	•	0	•	0	↕	1	4	00	001	111	
	RRA	[7 → 0] → CY	•	•	0	•	0	↕	1	4	00	011	111	
	RLC r	CY ← [7 ← 0] r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	2	8	11 00	001 000	011 r	Ⓔ
	RLC (HL)		↕	↕	0	P	0	↕	2	15	11 00	001 000	011 110	
	RLC (IX+d)		↕	↕	0	P	0	↕	4	23	11 11 ← 00	011 001 d 000	101 011 → 110	
	RLC (IY+d)		↕	↕	0	P	0	↕	4	23	11 11 ← 00	111 001 d 000	101 011 → 110	
	RL r	CY ← [7 ← 0] r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	2	8	11 00	001 010	011 r	Ⓔ
	RL (HL)		↕	↕	0	P	0	↕	2	15	11 00	001 010	011 110	
	RL (IX+d)		↕	↕	0	P	0	↕	4	23	11 11 ← 00	011 001 d 010	101 011 → 110	

命令群	ニモニック	オペレーション	S	Z	H	P/V	N	C	バイト	ステート	OP コード 76 543 210	
ローテート・シフト命令	RL (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 010 110	
	RRC r		↕	↕	0	P	0	↕	2	8	11 001 011 00 001 r	Ⓔ
	RRC (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 001 110	
	RRC (IX+d)	7 → 0 → CY r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 001 110	
	RRC (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 001 110	
	RR r		↕	↕	0	P	0	↕	2	8	11 001 011 00 011 r	Ⓔ
	RR (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 011 110	
	RR (IX+d)	7 → 0 → CY r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 011 110	
	RR (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 011 110	
	SLA r		↕	↕	0	P	0	↕	2	8	11 001 011 00 100 r	Ⓔ
	SLA (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 100 110	
	SLA (IX+d)	CY ← 7 ← 0 ← 0 r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 100 110	
	SLA (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 100 110	
	SRA r		↕	↕	0	P	0	↕	2	8	11 001 011 00 101 r	Ⓔ
	SRA (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 101 110	
	SRA (IX+d)	7 → 0 → CY r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 101 110	
	SRA (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 101 110	

命令群	ニモニック	オペレーション	フラグ						バイト	ステート	OP コード	
			S	Z	H	P/V	N	C			76 543 210	
ローテート・シフト命令	SRL r	0 → [7 → 0] → [CY] r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	2	8	11 001 011 00 111 r	Ⓔ
	SRL (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 111 110	
	SRL (IX+d)		↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 111 110	
	SRL (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 111 110	
	RLD	A [7–4 | 3–0] [7–4 | 3–0] (HL)	↕	↕	0	P	0	•	2	18	11 101 101 01 101 111	
	RRD	A [7–4 | 3–0] [7–4 | 3–0] (HL)	↕	↕	0	P	0	•	2	18	11 101 101 01 100 111	
ビット操作命令	BIT b, r	Z←$\overline{r_b}$	×	↕	1	×	0	•	2	8	11 001 011 01 b r	ⒺⒻ
	BIT b, (HL)	Z←$\overline{(HL)}_b$	×	↕	1	×	0	•	2	12	11 001 011 01 b 110	Ⓕ
	BIT b, (IX+d)	Z←$\overline{(IX+d)}_b$	×	↕	1	×	0	•	4	20	11 011 101 11 001 011 ← d → 01 b 110	Ⓕ
	BIT b, (IY+d)	Z←$\overline{(IY+d)}_b$	×	↕	1	×	0	•	4	20	11 111 101 11 001 011 ← d → 01 b 110	Ⓕ
	SET b, r	r_b←1	•	•	•	•	•	•	2	8	11 001 011 11 b r	ⒺⒻ
	SET b, (HL)	$(HL)_b$←1	•	•	•	•	•	•	2	15	11 001 011 11 b 110	Ⓕ
	SET b, (IX+d)	$(IX+d)_b$←1	•	•	•	•	•	•	4	23	11 011 101 11 001 011 ← d → 11 b 110	Ⓕ
	SET b, (IY+d)	$(IY+d)_b$←1	•	•	•	•	•	•	4	23	11 111 101 11 001 011 ← d → 11 b 110	Ⓕ
	RES b, r	r_b←0	•	•	•	•	•	•	2	8	11 001 011 10 b r	ⒺⒻ
	RES b, (HL)	$(HL)_b$←0	•	•	•	•	•	•	2	15	11 001 011 10 b 110	Ⓕ
	RES b, (IX+d)	$(IX+d)_b$←0	•	•	•	•	•	•	4	23	11 011 101 11 001 011 ← d → 10 b 110	Ⓕ
	RES b, (IY+d)	$(IY+d)_b$←0	•	•	•	•	•	•	4	23	11 111 101 11 001 011 ← d → 10 b 110	Ⓕ

命令群	ニモニック	オペレーション	S	Z	H	P/V	N	C	バイト	ステート	76	543	20	
			フラグ								OPコード			
ジャンプ・コール・リターン命令	JP nn	PC←nn	•	•	•	•	•	•	3	10	11 ← ←	000 n n	011 → →	
	JP cc, nn	If cc is true PC←nn otherwise continue	•	•	•	•	•	•	3	10	11 ← ←	cc n n	010 → →	Ⓖ
	JR e	PC←PC+e	•	•	•	•	•	•	2	12	00 ←	011 e−2	000 →	
	JR C, e	If C=0 continue If C=1 PC←PC+e	•	•	•	•	•	•	2	7 if C=0 12 if C=1	00 ←	111 e−2	000 →	
	JR NC, e	If C=1 continue If C=0 PC←PC+e	•	•	•	•	•	•	2	7 if C=1 12 if C=0	00 ←	110 e−2	000 →	
	JR Z, e	If Z=0 continue If Z=1 PC←PC+e	•	•	•	•	•	•	2	7 if Z=0 12 if Z=1	00 ←	101 e−2	000 →	
	JR NZ, e	If Z=1 continue If Z=0 PC←PC+e	•	•	•	•	•	•	2	7 if Z=1 12 if Z=0	00 ←	100 e−2	000 →	
	JP (HL)	PC←HL	•	•	•	•	•	•	1	4	11	101	001	
	JP (IX)	PC←IX	•	•	•	•	•	•	2	8	11 11	011 101	101 001	
	JP (IY)	PC←IY	•	•	•	•	•	•	2	8	11 11	111 101	101 001	
	DJNZ e	B←B−1 if B=0 continue if B≠0 PC←PC+e	•	•	•	•	•	•	2	8 if B=0 13 if B≠0	00 ←	010 e−2	000 →	
	CALL nn	(SP−1)←PC_H (SP−2)←PC_L PC←nn	•	•	•	•	•	•	3	17	11 ← ←	001 n n	101 → →	
	CALL cc, nn	If cc is false continue otherwise same as CALL nn	•	•	•	•	•	•	3	10 if cc is false 17 if cc is true	11 ← ←	cc n n	100 → →	Ⓖ
	RET	PC_L←(SP) PC_H←(SP+1)	•	•	•	•	•	•	1	10	11	001	001	
	RET cc	If cc is false continue otherwise same as RET	•	•	•	•	•	•	1	5 if cc is false 11 if cc is true	11	cc	000	Ⓖ
	RETI	Return from interrupt	•	•	•	•	•	•	2	14	11 01	101 001	101 101	
	RETN	Return from non maskable interrupt	•	•	•	•	•	•	2	14	11 01	101 000	101 101	
	RST p	(SP−1)←PC_H (SP−2)←PC_L PC_H←0 PC_L←p	•	•	•	•	•	•	1	11	11	t	111	Ⓗ
入出力命令	IN A, n	A←(n) A_{0−7}←n A_{8−15}←A	•	•	•	•	•	•	2	11	11 ←	011 n	011 →	
	IN r, (C)	r←(C) if r=110 only the flags will be affected A_{0−7}←C A_{8−15}←B	↕	↕	↕	P	0	•	2	12	11 01	101 r	101 000	Ⓔ
	INI	(HL)←(C) B←B−1 HL←HL+1 A_{0−7}←C A_{8−15}←B	×	③ ↕	×	×	1	•	2	16	11 10	101 100	101 010	

命令群	ニモニック	オペレーション	フラグ S	Z	H	P/V	N	C	バイト	ステート	OP コード 76	543	210	
	INIR	(HL)←(C) B←B−1 HL←HL+1 until B=0 A_{0-7}←C A_{8-15}←B	×	1	×	×	1	•	2	21 if B≠0 16 if B=0	11 10	101 110	101 010	
	IND	(HL)←(C) B←B−1 HL←HL−1 A_{0-7}←C A_{8-15}←B	×	③ ↕	×	×	1	•	2	16	11 10	101 101	101 010	
	INDR	(HL)←(C) B←B−1 HL←HL−1 until B=0 A_{0-7}←C A_{8-15}←B	×	1	×	×	1	•	2	21 if B≠0 16 if B=0	11 10	101 111	101 010	
	OUT n, A	(n)←A A_{0-7}←n A_{8-15}←A	•	•	•	•	•	•	2	11	11 ←	010 n	011 →	
	OUT (C), r	(C)←r A_{0-7}←C A_{8-15}←B	•	•	•	•	•	•	2	12	11 01	101 r	101 001	
入出力命令	OUTI	(C)←(HL) B←B−1 HL←HL+1 A_{0-7}←C A_{8-15}←B	×	③ ↕	×	×	1	•	2	16	11 10	101 100	101 011	Ⓔ
	OTIR	(C)←(HL) B←B−1 HL←HL+1 until B=0 A_{0-7}←C A_{8-15}←B	×	1	×	×	1	•	2	21 if B≠0 16 if B=0	11 10	101 110	101 011	
	OUTD	(C)←(HL) B←B−1 HL←HL−1 A_{0-7}←C A_{8-15}←B	×	③ ↕	×	×	1	•	2	16	11 10	101 101	101 011	
	OTDR	(C)←(HL) B←B−1 HL←HL−1 until B=0 A_{0-7}←C A_{8-15}←B	×	1	×	×	1	•	2	21 if B≠0 16 if B=0	11 10	101 111	101 011	

マシン語↔ニモニック対応表

00	NOP		40	LD	B, B	80	ADD	A, B	C0	RET	NZ
01	LD	BC, nn	41	LD	B, C	81	ADD	A, C	C1	POP	BC
02	LD	(BC), A	42	LD	B, D	82	ADD	A, D	C2	JP	NZ, nn
03	INC	BC	43	LD	B, E	83	ADD	A, E	C3	JP	nn
04	INC	B	44	LD	B, H	84	ADD	A, H	C4	CALL	NZ, nn
05	DEC	B	45	LD	B, L	85	ADD	A, L	C5	PUSH	BC
06	LD	B, n	46	LD	B, (HL)	86	ADD	A, (HL)	C6	ADD	A, n
07	RLCA		47	LD	B, A	87	ADD	A, A	C7	RST	00H
08	EX	AF, AF	48	LD	C, B	88	ADC	A, B	C8	RET	Z
09	ADD	HL, BC	49	LD	C, C	89	ADC	A, C	C9	RET	
0A	LD	A, (BC)	4A	LD	C, D	8A	ADC	A, D	CA	JP	Z, nn
0B	DEC	BC	4B	LD	C, E	8B	ADC	A, E	CB		
0C	INC	C	4C	LD	C, H	8C	ADC	A, H	CC	CALL	Z, nn
0D	DEC	C	4D	LD	C, L	8D	ADC	A, L	CD	CALL	nn
0E	LD	C, n	4E	LD	C, (HL)	8E	ADC	A, (HL)	CE	ADC	A, n
0F	RRCA		4F	LD	C, A	8F	ADC	A, A	CF	RST	08H
10	DJNZ	e	50	LD	D, B	90	SUB	B	D0	RET	NC
11	LD	DE, nn	51	LD	D, C	91	SUB	C	D1	POP	DE
12	LD	(DE), A	52	LD	D, D	92	SUB	D	D2	JP	NC, nn
13	INC	DE	53	LD	D, E	93	SUB	E	D3	OUT	n, A
14	INC	D	54	LD	D, H	94	SUB	H	D4	CALL	NC, nn
15	DEC	D	55	LD	D, L	95	SUB	L	D5	PUSH	DE
16	LD	D, n	56	LD	D, (HL)	96	SUB	(HL)	D6	SUB	n
17	RLA		57	LD	D, A	97	SUB	A	D7	RST	10H
18	JR	e	58	LD	E, B	98	SBC	A, B	D8	RET	C
19	ADD	HL, DE	59	LD	E, C	99	SBC	A, C	D9	EXX	
1A	LD	A, (DE)	5A	LD	E, D	9A	SBC	A, D	DA	JP	C, nn
1B	DEC	DE	5B	LD	E, E	9B	SBC	A, E	DB	IN	A, n
1C	INC	E	5C	LD	E, H	9C	SBC	A, H	DC	CALL	C, nn
1D	DEC	E	5D	LD	E, L	9D	SBC	A, L	DD		
1E	LD	E, n	5E	LD	E, (HL)	9E	SBC	A, (HL)	DE	SBC	A, n
1F	RRA		5F	LD	E, A	9F	SBC	A, A	DF	RST	18H
20	JR	NZ, e	60	LD	H, B	A0	AND	B	E0	RET	PO
21	LD	HL, nn	61	LD	H, C	A1	AND	C	E1	POP	HL
22	LD	(nn), HL	62	LD	H, D	A2	AND	D	E2	JP	PO, nn
23	INC	HL	63	LD	H, E	A3	AND	E	E3	EX	(SP), HL
24	INC	H	64	LD	H, H	A4	AND	H	E4	CALL	PO, nn
25	DEC	H	65	LD	H, L	A5	AND	L	E5	PUSH	HL
26	LD	H, n	66	LD	H, (HL)	A6	AND	(HL)	E6	AUD	n
27	DAA		67	LD	H, A	A7	AND	A	E7	RST	20H
28	JR	Z, e	68	LD	L, B	A8	XOR	B	E8	RET	PE
29	ADD	HL, HL	69	LD	L, C	A9	XOR	C	E9	JP	(HL)
2A	LD	HL, (nn)	6A	LD	L, D	AA	XOR	D	EA	JP	PE, nn
2B	DEC	HL	6B	LD	L, E	AB	XOR	E	EB	EX	DE, HL
2C	INC	L	6C	LD	L, H	AC	XOR	H	EC	CALL	PE, nn
2D	DEC	L	6D	LD	L, L	AD	XOR	L	ED		
2E	LD	L, n	6E	LD	L, (HL)	AE	XOR	(HL)	EE	XOR	n
2F	CPL		6F	LD	L, A	AF	XOR	A	EF	RST	28H
30	JR	NC, e	70	LD	(HL), B	B0	OR	B	F0	RET	P
31	LD	SP, nn	71	LD	(HL), C	B1	OR	C	F1	POP	AF
32	LD	(nn), A	72	LD	(HL), D	B2	OR	D	F2	JP	P, nn
33	INC	SP	73	LD	(HL), E	B3	OR	E	F3	DI	
34	INC	(HL)	74	LD	(HL), H	B4	OR	H	F4	CALL	P, nn
35	DEC	(HL)	75	LD	(HL), L	B5	OR	L	F5	PUSH	AF
36	LD	(HL), n	76	HALT		B6	OR	(HL)	F6	OR	n
37	SCF		77	LD	(HL), A	B7	OR	A	F7	RST	30H
38	JR	C, e	78	LD	A, B	B8	CP	B	F8	RET	M
39	ADD	HL, SP	79	LD	A, C	B9	CP	C	F9	LD	SP, HL
3A	LD	A, (nn)	7A	LD	A, D	BA	CP	D	FA	JP	M, nn
3B	DEC	SP	7B	LD	A, E	BB	CP	E	FB	EI	
3C	INC	A	7C	LD	A, H	BC	CP	H	FC	CALL	M, nn
3D	DEC	A	7D	LD	A, L	BD	CP	L	FD		
3E	LD	A, n	7E	LD	A, (HL)	BE	CP	(HL)	FE	CP	n
3F	CCF		7F	LD	A, A	BF	CP	A	FF	RST	38H

CB ××

00	RLC	B	40	BIT	0, B	80	RES	0, B	C0	SET	0, B
01	RLC	C	41	BIT	0, C	81	RES	0, C	C1	SET	0, C
02	RLC	D	42	BIT	0, D	82	RES	0, D	C2	SET	0, D
03	RLC	E	43	BIT	0, E	83	RES	0, E	C3	SET	0, E
04	RLC	H	44	BIT	0, H	84	RES	0, H	C4	SET	0, H
05	RLC	L	45	BIT	0, L	85	RES	0, L	C5	SET	0, L
06	RLC	(HL)	46	BIT	0, (HL)	86	RES	0, (HL)	C6	SET	0, (HL)
07	RLC	A	47	BIT	0, A	87	RES	0, A	C7	SET	0, A
08	RRC	B	48	BIT	1, B	88	RES	1, B	C8	SET	1, B
09	RRC	C	49	BIT	1, C	89	RES	1, C	C9	SET	1, C
0A	RRC	D	4A	BIT	1, D	8A	RES	1, D	CA	SET	1, D
0B	RRC	E	4B	BIT	1, E	8B	RES	1, E	CB	SET	1, E
0C	RRC	H	4C	BIT	1, H	8C	RES	1, H	CC	SET	1, H
0D	RRC	L	4D	BIT	1, L	8D	RES	1, L	CD	SET	1, L
0E	RRC	(HL)	4E	BIT	1, (HL)	8E	RES	1, (HL)	CE	SET	1, (HL)
0F	RRC	A	4F	BIT	1, A	8F	RES	1, A	CF	SET	1, A
10	RL	B	50	BIT	2, B	90	RES	2, B	D0	SET	2, B
11	RL	C	51	BIT	2, C	91	RES	2, C	D1	SET	2, C
12	RL	D	52	BIT	2, D	92	RES	2, D	D2	SET	2, D
13	RL	E	53	BIT	2, E	93	RES	2, E	D3	SET	2, E
14	RL	H	54	BIT	2, H	94	RES	2, H	D4	SET	2, H
15	RL	L	55	BIT	2, L	95	RES	2, L	D5	SET	2, L
16	RL	(HL)	56	BIT	2, (HL)	96	RES	2, (HL)	D6	SET	2, (HL)
17	RL	A	57	BIT	2, A	97	RES	2, A	D7	SET	2, A
18	RR	B	58	BIT	3, B	98	RES	3, B	D8	SET	3, B
19	RR	C	59	BIT	3, C	99	RES	3, C	D9	SET	3, C
1A	RR	D	5A	BIT	3, D	9A	RES	3, D	DA	SET	3, D
1B	RR	E	5B	BIT	3, E	9B	RES	3, E	DB	SET	3, E
1C	RR	H	5C	BIT	3, H	9C	RES	3, H	DC	SET	3, H
1D	RR	L	5D	BIT	3, L	9D	RES	3, L	DD	SET	3, L
1E	RR	(HL)	5E	BIT	3, (HL)	9E	RES	3, (HL)	DE	SET	3, (HL)
1F	RR	A	5F	BIT	3, A	9F	RES	3, A	DF	SET	3, A
20	SLA	B	60	BIT	4, B	A0	RES	4, B	E0	SET	4, B
21	SLA	C	61	BIT	4, C	A1	RES	4, C	E1	SET	4, C
22	SLA	D	62	BIT	4, D	A2	RES	4, D	E2	SET	4, D
23	SLA	E	63	BIT	4, E	A3	RES	4, E	E3	SET	4, E
24	SLA	H	64	BIT	4, H	A4	RES	4, H	E4	SET	4, H
25	SLA	L	65	BIT	4, L	A5	RES	4, L	E5	SET	4, L
26	SLA	(HL)	66	BIT	4, (HL)	A6	RES	4, (HL)	E6	SET	4, (HL)
27	SLA	A	67	BIT	4, A	A7	RES	4, A	E7	SET	4, A
28	SRA	B	68	BIT	5, B	A8	RES	5, B	E8	SET	5, B
29	SRA	C	69	BIT	5, C	A9	RES	5, C	E9	SET	5, C
2A	SRA	D	6A	BIT	5, D	AA	RES	5, D	EA	SET	5, D
2B	SRA	E	6B	BIT	5, E	AB	RES	5, E	EB	SET	5, E
2C	SRA	H	6C	BIT	5, H	AC	RES	5, H	EC	SET	5, H
2D	SRA	L	6D	BIT	5, L	AD	RES	5, L	ED	SET	5, L
2E	SRA	(HL)	6E	BIT	5, (HL)	AE	RES	5, (HL)	EE	SET	5, (HL)
2F	SRA	A	6F	BIT	5, A	AF	RES	5, A	EF	SET	5, A
30			70	BIT	6, B	B0	RES	6, B	F0	SET	6, B
31			71	BIT	6, C	B1	RES	6, C	F1	SET	6, C
32			72	BIT	6, D	B2	RES	6, D	F2	SET	6, D
33			73	BIT	6, E	B3	RES	6, E	F3	SET	6, E
34			74	BIT	6, H	B4	RES	6, H	F4	SET	6, H
35			75	BIT	6, L	B5	RES	6, L	F5	SET	6, L
36			76	BIT	6, (HL)	B6	RES	6, (HL)	F6	SET	6, (HL)
37			77	BIT	6, A	B7	RES	6, A	F7	SET	6, A
38	SRL	B	78	BIT	7, B	B8	RES	7, B	F8	SET	7, B
39	SRL	C	79	BIT	7, C	B9	RES	7, C	F9	SET	7, C
3A	SRL	D	7A	BIT	7, D	BA	RES	7, D	FA	SET	7, D
3B	SRL	E	7B	BIT	7, E	BB	RES	7, E	FB	SET	7, E
3C	SRL	H	7C	BIT	7, H	BC	RES	7, H	FC	SET	7, H
3D	SRL	L	7D	BIT	7, L	BD	RES	7, L	FD	SET	7, L
3E	SRL	(HL)	7E	BIT	7, (HL)	BE	RES	7, (HL)	FE	SET	7, (HL)
3F	SRL	A	7F	BIT	7, A	BF	RES	7, A	FF	SET	7, A

D D × ×				
09			ADD	IX, BC
19			ADD	IX, DE
21			LD	IX, nn
22			LD	(nn), IX
23			INC	IX
29			ADD	IX, IX
2A			LD	IX, (nn)
2B			DEC	IX
34			INC	(IX+d)
35			DEC	(IX+d)
36			LD	(IX+d), n
39			ADD	IX, SP
46			LD	B, (IX+d)
4E			LD	C, (IX+d)
56			LD	D, (IX+d)
5E			LD	E, (IX+d)
66			LD	H, (IX+d)
6E			LD	L, (IX+d)
70			LD	(IX+d), B
71			LD	(IX+d), C
72			LD	(IX+d), D
73			LD	(IX+d), E
74			LD	(IX+d), H
75			LD	(IX+d), L
77			LD	(IX+d), A
7E			LD	A, (IX+d)
86			ADD	A, (IX+d)
8E			ADC	A, (IX+d)
96			SUB	(IX+d)
9E			SBC	A, (IX+d)
A6			AND	(IX+d)
AE			XOR	(IX+d)
B6			OR	(IX+d)
BE			CP	(IX+d)
CB	d	06	RLC	(IX+d)
CB	d	0E	RRC	(IX+d)
CB	d	16	RL	(IX+d)
CB	d	1E	RR	(IX+d)
CB	d	26	SLA	(IX+d)
CB	d	2E	SRA	(IX+d)
CB	d	3E	SRL	(IX+d)
CB	d	46	BIT	0, (IX+d)
CB	d	4E	BIT	1, (IX+d)
CB	d	56	BIT	2, (IX+d)
CB	d	5E	BIT	3, (IX+d)
CB	d	66	BIT	4, (IX+d)
CB	d	6E	BIT	5, (IX+d)
CB	d	76	BIT	6, (IX+d)
CB	d	7E	BIT	7, (IX+d)
CB	d	86	RES	0, (IX+d)
CB	d	8E	RES	1, (IX+d)
CB	d	96	RES	2, (IX+d)
CB	d	9E	RES	3, (IX+d)
CB	d	A6	RES	4, (IX+d)
CB	d	AE	RES	5, (IX+d)
CB	d	B6	RES	6, (IX+d)
CB	d	BE	RES	7, (IX+d)
CB	d	C6	SET	0, (IX+d)
CB	d	CE	SET	1, (IX+d)
CB	d	D6	SET	2, (IX+d)
CB	d	DE	SET	3, (IX+d)
CB	d	E6	SET	4, (IX+d)
CB	d	EE	SET	5, (IX+d)
CB	d	F6	SET	6, (IX+d)
CB	d	FE	SET	7, (IX+d)
E1			POP	IX
E3			EX	(SP), IX
E5			PUSH	IX
E9			JP	(IX)
F9			LD	SP, IX

E D × ×		
40	IN	B, (C)
41	OUT	(C), B
42	SBC	HL, BC
43	LD	(nn), BC
44	NEG	
45	RETN	
46	IM	0
47	LD	I, A
48	IN	C, (C)
49	OUT	(C), C
4A	ADC	HL, BC
4B	LD	BC, (nn)
4D	RETI	
4F	LD	R, A
50	IN	D, (C)
51	OUT	(C), D
52	SBC	HL, DE
53	LD	(nn), DE
56	IM	1
57	LD	A, I
58	IN	E, (C)
59	OUT	(C), E
5A	ADC	HL, DE
5B	LD	DE, (nn)
5E	IM	2
5F	LD	A, R
60	IN	H, (C)
61	OUT	(C), H
62	SBC	HL, HL
67	RRD	
68	IN	L, (C)
69	OUT	(C), L
6A	ADC	HL, HL
6F	RLD	
72	SBC	HL, SP
73	LD	(nn), SP
78	IN	A, (C)
79	OUT	(C), A
7A	ADC	HL, SP
7B	LD	SP, (nn)
A0	LDI	
A1	CPI	
A2	INI	
A3	OUTI	
A8	LDD	
A9	CPD	
AA	IND	
AB	OUTD	
B0	LDIR	
B1	CPIR	
B2	INIR	
B3	OTIR	
B8	LDDR	
B9	CPDR	
BA	INDR	
BB	OTDR	

F D × ×				
09			ADD	IY, BC
19			ADD	IY, DE
21			LD	IY, nn
22			LD	(nn), IY
23			INC	IY
29			ADD	IY, IY
2A			LD	IY, (nn)
2B			DEC	IY
34			INC	(IY+d)
35			DEC	(IY+d)
36			LD	(IY+d), n
39			ADD	IY, SP
46			LD	B, (IY+d)
4E			LD	C, (IY+d)
56			LD	D, (IY+d)
5E			LD	E, (IY+d)
66			LD	H, (IY+d)
6E			LD	L, (IY+d)
70			LD	(IY+d), B
71			LD	(IY+d), C
72			LD	(IY+d), D
73			LD	(IY+d), E
74			LD	(IY+d), H
75			LD	(IY+d), L
77			LD	(IY+d), A
7E			LD	A, (IY+d)
86			ADD	A, (IY+d)
8E			ADC	A, (IY+d)
96			SUB	(IY+d)
9E			SBC	A, (IY+d)
A6			AND	(IY+d)
AE			XOR	(IY+d)
B6			OR	(IY+d)
BE			CP	(IY+d)
CB	d	06	RLC	(IY+d)
CB	d	0E	RRC	(IY+d)
CB	d	16	RL	(IY+d)
CB	d	1E	RR	(IY+d)
CB	d	26	SLA	(IY+d)
CB	d	2E	SRA	(IY+d)
CB	d	3E	SRL	(IY+d)
CB	d	46	BIT	0, (IY+d)
CB	d	4E	BIT	1, (IY+d)
CB	d	56	BIT	2, (IY+d)
CB	d	5E	BIT	3, (IY+d)
CB	d	66	BIT	4, (IY+d)
CB	d	6E	BIT	5, (IY+d)
CB	d	76	BIT	6, (IY+d)
CB	d	7E	BIT	7, (IY+d)
CB	d	86	RES	0, (IY+d)
CB	d	8E	RES	1, (IY+d)
CB	d	96	RES	2, (IY+d)
CB	d	9E	RES	3, (IY+d)
CB	d	A6	RES	4, (IY+d)
CB	d	AE	RES	5, (IY+d)
CB	d	B6	RES	6, (IY+d)
CB	d	BE	RES	7, (IY+d)
CB	d	C6	SET	0, (IY+d)
CB	d	CE	SET	1, (IY+d)
CB	d	D6	SET	2, (IY+d)
CB	d	DE	SET	3, (IY+d)
CB	d	E6	SET	4, (IY+d)
CB	d	EE	SET	5, (IY+d)
CB	d	F6	SET	6, (IY+d)
CB	d	FE	SET	7, (IY+d)
E1			POP	IY
E3			EX	(SP), IY
E5			PUSH	IY
E9			JP	(IY)
F9			LD	SP, IY

ニモニック↔マシン語対照表

8ビット・ロード

×	I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn)	n
LD A, ×	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
LD B, ×			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
LD C, ×			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
LD D, ×			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
LD E, ×			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
LD H, ×			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
LD L, ×			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
LD (HL), ×			77	70	71	72	73	74	75							36 n
LD (BC), ×			02													
LD (DE), ×			12													
LD (IX+d), ×			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
LD (IY+d), ×			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
LD (nn), ×			32 n n													
LD I, ×			ED 47													
LD R, ×			ED 4F													

16ビット・ロード

×	AF	BC	DE	HL	SP	IX	IY	nn	(nn)
LD AF, ×									
LD BC, ×								01 n n	ED 4B n n
LD DE, ×								11 n n	ED 5B n n
LD HL, ×								21 n n	2A n n
LD SP, ×				F9		DD F9	FD F9	31 n n	ED 7B n n
LD IX, ×								DD 21 n n	DD 2A n n
LD IY, ×								FD 21 n n	FD 2A n n
LD (nn), ×		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n		
PUSH ×	F5	C5	D5	E5		DD E5	FD E5		
POP ×	F1	C1	D1	E1		DD E1	FD E1		

ブロック・転送

LDI	ED A0
LDIR	ED B0
LDD	ED A8
LDDR	ED B8

ブロック・サーチ

CPI	ED A1
CPIR	ED B1
CPD	ED A9
CPDR	ED B9

8ビット算術論理演算

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
ADD A, ×	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADC A, ×	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUB ×	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SBC A, ×	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
AND ×	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
XOR ×	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
OR ×	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
CP ×	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INC ×	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DEC ×	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

CPUコントロール

NOP	00
HALT	76
DI	F3
EI	FB
IM 0	ED 46
IM 1	ED 56
IM 2	ED 5E

16ビット算術演算

×	BC	DE	HL	SP	IX	IY
ADD HL, ×	09	19	29	39		
ADD IX, ×	DD 09	DD 19		DD 39	DD 29	
ADD IY, ×	FD 09	FD 19		FD 39		FD 29
ADC HL, ×	ED 4A	ED 5A	ED 6A	ED 7A		
SBC HL, ×	ED 42	ED 52	ED 62	ED 72		
INC ×	03	13	23	33	DD 23	FD 23
DEC ×	0B	1B	2B	3B	DD 2B	FD 2B

エクスチェンジ

EX AF, AF'	08
EX DE, HL	EB
EX (SP), HL	E3
EX (SP), IX	DD E3
EX (SP), IY	FD E3
EXX	D9

アキュムレータ操作

DAA	27
CPL	2F
NEG	ED 44
CCF	3F
SCF	37

ローテート，シフト

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
RLC ×	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
RRC ×	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
RL ×	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
RR ×	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
SLA ×	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
SRA ×	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
SRL ×	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
RLD								ED 6F		
RRD								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F

ジャンプ，コール，リターン

×	UN COND	C	NC	Z	NZ	PE	PO	M	P	
JP ×, nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JP ×, e	18 e−2	38 e−2	30 e−2	28 e−2	20 e−2					
JP (HL)	E9									
JP (IX)	DD E9									
JP (IY)	FD E9									
CALL ×, nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DJNZ e										10 e−2
RET ×	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETI	ED 4D									
RETN	ED 45									

リスタート

RST 00H	C7
RST 08H	CF
RST 10H	D7
RST 18H	DF
RST 20H	E7
RST 28H	EF
RST 30H	F7
RST 38H	FF

ビット操作

×＼	A	B	C	D	E	H	L	(HL)	(IX +d)	(IY +d)
BIT 0, ×	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
BIT 1, ×	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
BIT 2, ×	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
BIT 3, ×	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
BIT 4, ×	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
BIT 5, ×	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
BIT 6, ×	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
BIT 7, ×	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RES 0, ×	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
RES 1, ×	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
RES 2, ×	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
RES 3, ×	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
RES 4, ×	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
RES 5, ×	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
RES 6, ×	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
RES 7, ×	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET 0, ×	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
SET 1, ×	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
SET 2, ×	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
SET 3, ×	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
SET 4, ×	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
SET 5, ×	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
SET 6, ×	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
SET 7, ×	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

入　力

IN A, n	DB n
IN A, (C)	ED 78
IN B, (C)	ED 40
IN C, (C)	ED 48
IN D, (C)	ED 50
IN E, (C)	ED 58
IN H, (C)	ED 60
IN L, (C)	ED 68
INI	ED A2
INIR	ED B2
IND	ED AA
INDR	ED BA

出　力

OUT n, A	D3 n
OUT (C), A	ED 79
OUT (C), B	ED 41
OUT (C), C	ED 49
OUT (C), D	ED 51
OUT (C), E	E 59
OUT (C), H	ED 61
OUT (C), L	ED 69
OUTI	ED A3
OTIR	ED B3
OUTD	ED AB
OTDR	ED BB

アスキーコード一覧表

上位4ビット→

下位4ビット↓	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		DE		0	@	P	、	p				ー	タ	ミ		
1	SH	DI	!	1	A	Q	a	q			。	ア	チ	ム		円
2	SX	D2	"	2	B	R	b	r			「	イ	ツ	メ		年
3	EX	D3	#	3	C	S	c	s			」	ウ	テ	モ		月
4	ET	D4	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	EQ	NK	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	AK	SN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	BL	EB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	BS	CN	(	8	H	X	h	x			ィ	ク	ネ	リ	♠	
9	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	♥	
A	LF	SB	*	:	J	Z	j	z			ェ	コ	ハ	レ	♦	
B	HM	EC	+	;	K	[	k	{			ォ	サ	ヒ	ロ	♣	
C	CL	→	,	<	L	¥	l	¦			ャ	シ	フ	ワ	●	
D	CR	←	−	=	M	]	m	}			ュ	ス	ヘ	ン	○	
E	SO	↑	.	>	N	^	n	~			ョ	セ	ホ	゛		
F	SI	↓	/	?	O	_	o				ッ	ソ	マ	゜		

〈著者紹介〉

日高　徹

1949年　栃木県宇都宮市生まれ
早稲田大学商学部卒業後、商社やカー用品メーカーに十数年勤務。現在はフリーのゲームデザイナー。
市販された作品に『ホーンテッドケイブ』『マジックガーデン』『北斗の拳』『ガンダーラ』など。
著書は、88版／98版『マシン語ゲームプログラミング』88版『マシン語サウンド・プログラミング』(以上アスキー)、88版／98版『マシン語ゲームグラフィックス』(小学館)、88版／98版『はじめてのマシン語』『8086マシン語秘伝の書』(以上啓学出版)。
趣味はトレーニング。剣道三段。運転免許証を全種類持つ隠れプロドライバーでもある。

[編集部からのお願い]
本書の内容に関する質問等は書面にて当編集部宛へお送り願います。
電話による質問等にはいっさい応じられません。

Z80 マシン語秘伝の書

1989年 8 月15日　第 1 刷発行
1991年 8 月10日　第 6 刷発行

著者　日高(ひだか)　徹(とおる)

発行所　啓学出版株式会社
代表者　三井数美

郵便番号　101
東京都千代田区神田神保町1-46
振替　東京　3-109286
電話　東京03(3233)3731[編集部]
　　　東京03(3233)3795[販売部]

印刷／昭和工業写真印刷所
製本／徳住製本所

ISBN 4-7665-1038-0
本書の定価はカバーに表示してあります

Printed in Japan
Tsuchi

ISBN4-7665-1038-0 C3055 P1800E
定価1,800円（本体1,748円）

Z80 マシン語秘伝の書



(出) 1953

